D2.1 Architecture for Attribute-based Credential Technologies – Version 1

Jan Camenisch, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Kai Rannenberg, Harald Zwingelberg

Editor:	*Ioannis Krontiris (Goethe University Frankfurt)*
Reviewers:	*Norbert Götze (Nokia Siemens Networks), Jakob Illeborg Pagter (Alexandra Institute)*
Identifier:	*D2.1*
Type:	*Deliverable*
Version:	*1.1*
Date:	*01/12/2011*
Status:	*Final*
Class:	*Public*

Abstract

The goal of ABC4Trust is to address the federation and interchangeability of technologies that support trustworthy yet privacy-preserving Attribute-based Credentials (ABC). Towards this goal, one of the main objectives of the project is to define a common, unified architecture for privacy-ABC systems to allow comparing their respective features and combining them on common platforms. The first version of this architecture is described in the deliverable at hand. Its main contribution is the specification of the data artifacts exchanged between the implicated entities (i.e. issuer, user, verifier, revocation authority, etc.), in such a way that the underlying differences of concrete Privacy-ABC implementations are abstracted away through the definition of formats that can convey information independently from the mechanism-specific cryptographic data. It also defines all technology-agnostic components and corresponding APIs a system needs to implement in order to perform the corresponding operations, i.e. to process an obtained issuance/presentation policy, perform the selection of applicable credentials for a given policy or to trigger the mechanism-specific generation of the cryptographic evidence. How Privacy-ABCs can be applied in existing identity protocols and frameworks such as WS-*, SAML, OpenID, OAuth and X.509 and how Privacy-ABCs can help to alleviate some of the security, privacy and scalability issues of these protocols is also discussed.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Members of the ABC4TRUST consortium

1.	Alexandra Institute AS	ALX	Denmark
2.	CryptoExperts SAS	CRX	France
3.	Eurodocs AB	EDOC	Sweden
4.	IBM Research – Zurich	IBM	Switzerland
5.	Johann Wolfgang Goethe – Universität Frankfurt	GUF	Germany
6.	Microsoft Research and Development	MS	France
7.	Miracle A/S	MCL	Denmark
8.	Nokia-Siemens Networks GmbH & Co. KG	NSN	Germany
9.	Research Academic Computer Technology Institute	CTI	Greece
10.	Söderhamn Kommun	SK	Sweden
11.	Technische Universität Darmstadt	TUD	Germany
12.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany

List of Contributors

Chapter	**Author(s)**
Executive Summary	Ioannis Krontiris (GUF)
First Chapter	Ioannis Krontiris (GUF), Kai Rannenberg (GUF)
Second Chapter	Gregory Neven (IBM)
Third Chapter	Jan Camenisch (IBM), Anja Lehmann (IBM)
Fourth Chapter	Jan Camenisch (IBM), Anja Lehmann (IBM), Gregory Neven (IBM), Christian Paquin (MS)
Fifth Chapter	Jan Camenisch (IBM), Anja Lehmann (IBM), Gregory Neven (IBM), Christian Paquin (MS)
Sixth Chapter	Harald Zwingelberg (ULD)
Seventh Chapter	Christian Paquin (MS)

Executive Summary

This deliverable presents the first version of the ABC4Trust architecture. It produces an architectural framework for Privacy-ABC technologies that allows different realizations of these technologies to coexist, be interchanged, and federated. This enables users to obtain credentials following different Privacy-ABC technologies and use them indifferently on the same hardware and software platforms, as well as service providers to adopt whatever Privacy-ABC technology best suits their needs.

In particular, the architecture has been designed to decompose future (reference) implementations of Privacy-ABC technologies into sets of modules and specify the abstract functionality of these components in such a way that they are independent from algorithms or cryptographic components used underneath. The functional decomposition foresees possible architectural extensions to additional functional modules that may be desirable and feasible using future Privacy-ABC technologies or extensions of existing ones.

The architecture of ABC4Trust is defined by following a layered approach, where all Privacy-ABC related functionalities are grouped together in a layer called ABCE (ABC Engine). It provides simple interfaces towards the application layer, thereby abstracting the internal design and structure. So the deliverable defines and standardizes all the technology-agnostic components of the ABCE layer, as well as the APIs they provide. For the latter, the deliverable defines first the interfaces the ABCE components offer to the upper layers (e.g. Application), as well as the APIs that the different components within the ABCE layer expose to each other.

Equally important in the architecture is the specification of the data artifacts exchanged between the implicated actors, in such a way that the underlying differences of concrete Privacy-ABCs are abstracted away through the definition of formats that can convey information independently from the mechanism-specific cryptographic data. So the document emphasizes on the XML based specification of the corresponding messages exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials.

Consideration is also given on how the proposed architecture integrates with existing identity management systems. The deliverable provides an analysis regarding the applicability of the ABC4Trust architecture to the popular existing identity protocols and frameworks such as WS-*, SAML, OpenID, OAuth and X.509. It shows how Privacy-ABCs can be applied in these protocols and how the former can help to alleviate some of their security, privacy and scalability issues of the latter.

Finally, an overview on legal considerations of deploying Privacy-ABCs is also provided. It describes how the benefits of Privacy-ABCs relate to privacy requirements from the legal point of view and lays the foundation for evaluating the legal obligations and rights between the implicated parties. This analysis will later be used to perform a legal evaluation on the specific pilot scenarios applying the architecture.

Within the ABC4Trust project, WP4 will use the input of this document to provide a reference implementation of the architecture, which will be later used for realizing the pilot scenarios of WP6 and WP7. Also WP3 will use the architecture design presented here to perform a comparison of features at common levels (cryptographic properties, security tokens, etc.).

The deployment of the reference implementation of this architecture in the pilot scenarios during the next months will give valuable feedback to the architecture design and the experiences gained will enable its finalization in the second version (M39). The second version will also concentrate on more detailed definitions needed for advanced features (e.g. algebraic operation in predicates or in carry-over issuing, efficient updates of attributes, limited spending, inspection of proofs, etc.)

Table of Contents

Index of Figures

Index of Tables

1 Introduction

Many electronic applications and services require some authentication of participants to establish trust relations, either for only one endpoint of communication or for both. One widely used mechanism for this is password-based authentication. Today, individuals are asked to maintain dozens of different usernames and passwords, one for each website with which they interact. This authentication mechanism is not always optimal, since it creates a burden to individuals and encourages the reuse of passwords through multiple services, which in turns makes online fraud and identity theft easier. Spoofed website, stolen passwords and compromised accounts have negative impact not only to individuals but also to businesses and governments, who are unable to offer high-value online services.

Given the weaknesses of such a simple authentication method, multiple alternate techniques have been developed to provide a higher degree of access control and personal data management. Identity management (or IdM for short) consists of the processes and all underlying technologies for the creation, management and usage of digital identities. Broadly speaking, these techniques come to cover the need of individuals and businesses to verify whether a presented identifier or identity is actually representing the entity one trusts or that is entitled to enter into a certain transaction or communication [DEF11].

In this chapter, we start with a brief discussion about the privacy issues concerning the current IdM systems. Then we introduce Attribute-based credentials and how they can be used to effectively resolve these privacy issues. Following that, we go through the objectives and the goals, which ABC4Trust project is aiming for, and continue with describing the design decisions and strategies which have been considered in the ABC4Trust architectural work. The last section provides an overview of the document structure and the organization of the following chapters.

1.1 Privacy issues of current IdM systems

In their everyday offline transactions, people have to present credentials in order to perform a number of operations. There are several aspects in these transactions that are privacy respecting, but have not been preserved in similar transactions online. For example, when individuals have to present their ID card to open a bank account or board an airplane, the government authority issuing the ID cards does not learn about every place individuals have to present their card.

On the other hand however, there are also some aspects of offline transactions that are not privacy respecting. For example, showing the ID card to buy alcohol at the store reveals extraneous information, such as the exact date of birth, while what is actually requested is to prove that one is over a certain age. This is not really a problem in the offline world, because the infrastructure (i.e., the clerk behind the counter) is not equipped to log and remember all disclosed information; but things change in the online world: disclosed information is forever remembered.

Users' online privacy is increasingly threatened as a number of countries are introducing or are about to introduce electronic identity cards (eID) and drivers licenses [FP11], expanding the use of credentials in the online world. Moreover, electronic ticketing and toll systems are also widely used all over the world. As such electronic devices become widespread for identification, authentication, and payment (which links them to people through credit card systems) in a broad range of scenarios.

One desirable goal of building online identity management systems should be to keep the privacy respecting aspects of the offline paradigm and resolve the negative aspects. To see the problems that emerge for privacy, one has to observe the flow and storage of information between the involved entities. A typical identity management is based on the relations between three core parties: the user

(U) who requests a service from the Relying Party (RP) that offers the service and relies on the Identity Service Provider (IdSP) to provide authentic information about the identity of the user.

We briefly review below some of the main privacy aspects of the practices followed today, based on the parties and the relations introduced [Bra00].

1.1.1 IdSP knows all user transactions

Even though the IdSP does not necessarily need to know where the user is authenticating and which service she is requesting for, this happens in a large portion of the existing federation identity management systems. More specifically, in those systems where the Identity Service Provider is involved each time a user authenticates to a Relying Party, the IdSP might be able to keep track of the user actions, depending on the exchanged information between the IdSP and the RP. This enables the IdSP to trace and link all communications and transactions of each user. Moreover, if the user does not perform an active role in the information exchange between the IdSP and the RP (e.g. OpenID [OpenID2.0]), there is a high security risk of identity theft and impersonation of the user by the IdSP or an intruder who has gained access to the IdSP resources.

1.1.2 Linkability across domains

In today’s identity management systems, each Relying Party can store all the tokens that are presented to it, and can link them together. The simplest example is X.509 certificates where the certificate’s public key and issuer’s signature act as kind of digital fingerprint, inescapably leaving a digital trail wherever the citizen presents the certificate. In this manner, dossiers can automatically be compiled for each individual about his or her habits, behavior, movements, preferences, characteristics, and so on. Different Relying Parties can exchange and link their data on the same basis.

1.1.3 Proportionality often violated

There are many scenarios where the use of certificates unnecessarily reveals the identity of their holder, for instance scenarios where a service platform only needs to verify the age of a user but not his/her actual identity.

A typical example is the citizen PKI, where each citizen is provided with a X.509 certificate [X509] as the digital identifier for securely accessing the online services offered by the governments. These certificates contain a set of attributes such as full name, date of birth, gender, and ID number, and inevitably all will be revealed to the Relaying Party each time the certificate is presented.

Revealing more information than necessary not only harms the privacy of the users, but also increases the risk of abuse of information such as identify theft when information revealed falls in the wrong hands.

1.1.4 Privacy Attribute-based Credentials

Over the past years, a number of technologies have been developed to build Privacy-enhanced Attribute-based Credential (Privacy-ABC) systems in a way that they can be trusted, like normal cryptographic certificates, while at the same time they protect the privacy of their holder, resolving the problems discussed in the previous section, in addition to other properties.

Such Privacy-ABCs are issued just like ordinary cryptographic credentials (e.g., X.509 credentials) using a digital (secret) signature key. However, as we will see in Chapter 2, Privacy-ABCs allow their holder to transform them into a new token, in such a way that the privacy of the user is protected. Still, these transformed tokens can be verified just like ordinary cryptographic credentials (using the public verification key of the issuer) and offer the same strong security.

There are a handful of proposals on how to realize a Privacy-ABC system in the literature [Cha85, Bra93, CL01, CL04]. Notable is especially the appearance of two technologies, IBM's Identity Mixer and Microsoft's U-Prove, as well as extended work done in past EU projects. In particular, the EU-funded projects PRIME[1] and PrimeLife[2] have actually shown that the state-of-the art research prototypes of Privacy-ABC systems can indeed confront the privacy challenges of identity management systems.

The PRIME project has designed an architecture for privacy-enhancing identity management that combines anonymous credentials with attribute-based access control, and anonymous communication. That project has further demonstrated the practical feasibility with a prototypical implementation of that architecture and demonstrators for application areas such as e-learning and location-based services. PRIME has, however, also uncovered that in order for these concepts to be applicable in practice further research is needed in the areas of user interfaces, policy languages, and infrastructures.

The PrimeLife project has set out in 2008 to take up these challenges and made successful steps towards solutions in these areas. For instance, it has shown that Privacy-ABC systems can be employed on Smart Cards and thus address the requirements of privacy-protecting eID cards [BCGS09]. Also, in the last decade, a large number of research papers have been published solving probably all roadblocks to employ Privacy-ABC technologies in practice. This includes means to revoke certificate [Ngu05, BDDD07, CL02, CKS09], protection of credentials from malware [Cam06], protection against credential abuse [CHK+06, CHL06], proving properties about certified attributes [CG08, CCS08], and means to revoke anonymity in case of misuse [CS03].

Despite all of this, the effort of understanding Privacy-ABC technologies so-far was rather theoretical and limited to individual research prototypes. Indeed, PRIME and PrimeLife showed that Privacy-ABC technologies provide the desirable level of privacy protection, but so far this has been demonstrated in a very limited number of actual production environments with real users.

Furthermore, there are no commonly agreed set of functions, features, formats, protocols, and metrics to gauge and compare these Privacy-ABC technologies, and it is hard to judge the pros and cons of the different technologies to understand which ones are best suited to which scenarios.

Thus, there is still a gap between the technical cryptography and protocol sides of these technologies and the reality of deploying them in production environments. A related problem with these emerging technologies is the lack of standards to deploy them. For instance, a position paper published by ENISA on "Privacy Features of European eID Card Specifications" [ENISA09] observes that Privacy-ABC "technologies have been available for a long time, but there has not been much adoption in mainstream applications and eID card applications" even though countries such as Austria and Germany have taken some important steps in this sense.

1.2 The ABC4Trust Project

The aim of the ABC4Trust project is to deepen the understanding in Privacy-ABC technologies, enable their efficient and effective deployment in practice, and their federation in different domains. To this end, the project:

1. Produces an architectural framework for Privacy-ABC technologies that allows different realizations of these technologies to coexist, be interchanged, and federated

    a. Identify and describe the different functional components of Privacy-ABC technologies, e.g. for request and issue of credentials and presentation of tokens;

---

1 www.prime-project.eu
2 www.primelife.eu

   b. Produce a specification of data formats, interfaces, and protocols formats for this framework;

2. Defines criteria to compare the properties of realizations of these components in different technologies; and

3. Provides reference implementations of each of these components.

With a comparative understanding of today's available Privacy-ABC technologies, it will be easier for different user communities to decide which technology best serves them in which application scenario. It will also be easier to migrate to newer Privacy-ABC technologies that will undoubtedly appear over time. In addition, the same users may want to access applications requiring different Privacy-ABC technologies, and the same applications may want to cater to user communities preferring different Privacy-ABC technologies. Finally, the architecture and protocol specifications proposed by the ABC4Trust project flatten the road towards establishing standards that allow for the interchangeability and federation of Privacy-ABC technologies.

1.3 The ABC4Trust Architecture

From the three project goals above, this document focuses on the first one, namely the architecture for Privacy-ABC technologies. This is presented extensively in the chapters to follow. It is however useful for the reader to first understand the goals and design considerations that were taken into account during the design of this architecture. This subsection elaborates on these decisions and prepares the reader for the chapters that follow.

1.3.1 Goals of the Architecture

A contribution of this project to the state of the art is the definition of a common unified architecture for federating and interchanging different Privacy-ABC systems in a way that

1. users will be able to obtain credentials for many Privacy-ABC technologies and use them indifferently on the same hardware and software platforms, and

2. service providers and IdSPs will be able to adopt whatever Privacy-ABC technology best suits their needs.

Furthermore, the architecture has been designed to decompose future (reference) implementations of Privacy-ABC technologies into sets of modules and specify the abstract functionality of these components in such a way that they are independent from algorithms or cryptographic components used underneath. The functional decomposition foresees possible architectural extensions to additional functional modules that may be desirable and feasible using future Privacy-ABC technologies or extensions of existing ones.

Indeed the project aims not only to federate Privacy-ABC systems but to let them coexist on the same platform. This in turn implies that different systems must be able to share common architecture elements such as user interfaces or credential stores. Thus common APIs must be enforced across different Privacy-ABC implementations to ensure their possible coexistence and interchangeability on the same network node. Similarly, different systems should use common communication wrappers to encode and exchange tokens and other items when communicating with peers on different network node, so that a token recipient can immediately determine what Privacy-ABC technology the token pertains to.

Thus the architectural framework specifies unified data formats and protocols across Privacy-ABC implementations to enable not just coexistence and interchangeability on the same network node but also coexistence and possible federation across different network nodes.

1.3.2 Architectural Strategies

This section describes the design decisions and strategies that affect the overall organization of the architecture and its higher-level structures.

1.3.2.1 A Layered Approach

The architecture of ABC4Trust is defined by following a layered approach, where all Privacy-ABC related functionalities are grouped together in a layer called ABCE (ABC Engine). It provides simple interfaces towards the application layer, thereby abstracting the internal design and structure. So, the focus of the ABC4Trust architecture is to define and standardise the ABCE layer and its interfaces to the upper layers (e.g. Application). With this respect, it does not analyze the internals of the other layers, but it only concentrates on defining the interfaces necessary for those layers to use the functionality of the ABCE and incorporate the corresponding tokens in the overall system.

Equally important in the architecture is the specification of the data artifacts exchanged between the implicated entities, in such a way that the underlying differences of concrete Privacy-ABCs are abstracted away through the definition of formats that can convey information independently from the mechanism-specific cryptographic data.

In particular, this document concentrates in the following aspects:

- *Components and interfaces* – We define all technology-agnostic components and corresponding methods of the ABCE layer (see Chapter 3). That is, it contains e.g. the components responsible to parse an obtained presentation policy, perform the selection of applicable credentials for a given policy or to trigger the mechanism-specific generation of the cryptographic evidence. These components and their functionality are described considering separately the various phases of identity management protocols, i.e. presentation of a token, issuance of a credential, inspection and revocation.

  For each of these phases, the description of the implicated ABCE components is completed by referencing also the corresponding interfaces they offer. There are two kinds of interfaces specifications defined in this document (see Chapter 5). The first one concerns the top-level application programming interfaces (API) that the ABCE layer exposes to the upper layers, i.e. the application layer, so that developers can build easily ABC-enabled applications. The second one concerns the APIs that the different components within the ABCE layer expose to each other, and how they collaborate in a typical flow of invocations. This is mainly intended for developers who want to build new cryptographic providers and plug them into the framework.
- *Data specification* – The issuance and presentation of Privacy-ABC credentials are interactive processes, potentially involving multiple exchanges of messages. Chapter 4 defines the contents, encoding and processing of these messages. In particular, it specifies the data artifacts exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials. Note that the document remains generic on which specific protocols are used to issue or present Privacy-ABC credentials. There are several existing messaging protocols, in which these credentials can be embedded, or new ones could be defined in the future.

1.3.2.2 Privacy by design

Privacy-ABCs and the architecture presented in this document are based on the necessity to rethink the way systems collect and process personal data and design them in such a way that privacy aspects are taken into account from the beginning. A central aspect and aim should be to reduce amount and duration of the processing of personal data to the necessary minimum. Chapter 6 elaborates on how Privacy-ABCs assist such efforts and may even make it possible to reassess of what is considered necessary processing today. To understand and evaluate the legal obligations and rights between the parties it is necessary to have an overview over the legal relation between the entities. To lay a foundation for future assessment within the project chapter 6 describes the legal relations between Issuer, User and Verifier.

1.3.2.3 Building Privacy-ABC-enabled applications

ABC4Trust targets to provide an open reference implementation of the architecture described in this document as part of its upcoming contributions. The reference implementation of ABC4Trust will be embedded into example applications showing how to integrate the reference components into a sample client-server platform.

Application developers can integrate the reference implementation of the ABC4Trust architecture directly in their applications, without having to know how its layers are internally structured. That means the application can incorporate user authentication functionality using Privacy-ABC, according to the access policy of the requested service, by executing directly the necessary protocols for policy and token exchange. For that, the application developers must simply follow the interfaces and data formats described in this document.

However, other approaches are also possible. For example, following a passive federated redirection pattern, the application may redirect the user to a Relying Party Secure Token Service (RP-STS) component for authentication. This is shown and discussed in more details in Chapter 7, where we discuss how the ABC4Trust architecture can be integrated with existing federated systems.

1.4 Structure of the document

The rest of this document is organized as follows:

Chapter 2 gives an introduction to the features supported by Privacy-ABCs and the actors involved, in different kind of interactions, namely the presentation of tokens, inspection, credential issuance and revocation.

Chapter 3 presents the modules of the ABC4Trust architecture and concentrates in particular on the ABCE layer. It revisits each of the scenarios introduced in Chapter 2 and shows specifically how they are performed by the ABCE modules

Chapter 4 can be considered as the core part of this document. It provides the specification for data artifacts exchanged during the Privacy-ABCs lifecycle (issuance, presentation, revocation, and inspection). It introduces an XML notation to specific all the necessary data formats, e.g. credentials contents, access policies, presentation tokens, etc., as well as their wrapper message formats.

Chapter 5 defines the APIs for each of the ABCE modules. It separates between two cases: first the interfaces exposed to the outside (and in particular to the application layer) and the interfaces exposed internally between the ABCE modules.

Chapter 6 elaborates on legal goals and principles that should be taken under consideration not only in the design of identity management systems but also in the establishment of a methodology for their threat analysis.

Chapter 7 presents an overview of the most popular identity protocols and frameworks (e.g. WS-*, SAML, OpenID, OAuth, and X.509) and describes the common challenges of these federated systems concerning security, privacy and scalability. The analysis provided in this chapter, demonstrate how Privacy-ABC technologies can help to alleviate these challenges. The reader may note that in this chapter the Identity Service Provider is named “Identity Provider”. The reason for this is, that many of the existing protocols use this term, though it is misleading, as the respective entity does not provide identities.

2 Features and Concepts of Privacy-ABCs

This section provides a detailed explanation on the features supported by privacy-enhancing attribute-based credentials (Privacy-ABCs), on the different involved entities, and on the type of interactions that they engage in.

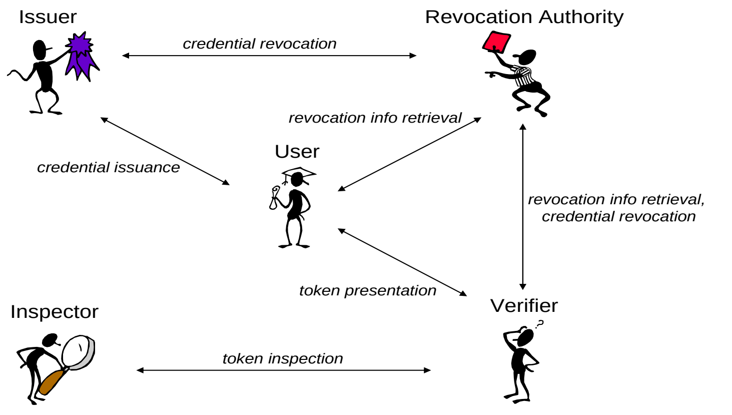

Figure 2.1 - Entities and interactions diagram

Figure 2.1 gives an overview of the different entities and the interactions between them.

- The *User* is at the center of the picture, collecting credentials from various Issuers and controlling which information from which credentials she presents to which verifiers. The human User is represented by her *User Agent*, a software component running either on a local device (e.g., on the User's computer or mobile phone) or remotely on a trusted cloud service. The User may own special hardware tokens to which credentials can be bound to improve security. In the identity management literature, the User is sometimes referred to as the requestor or the subject.
- An *Issuer* issues credentials to Users, thereby vouching for the correctness of the information contained in the credential with respect to the User to whom the credential is issued. Before issuing a credential, the Issuer may have to authenticate the User, which it may do using Privacy-ABCs, using a different online mechanism (e.g., username and password), or using out-of-band communication (e.g., by requiring the User to physically present herself at the Issuer's office). In the identity management literature, the Issuer is sometimes referred to as the identity provider or attribute authority.
- A *Verifier* protects access to a resource or service that it offers by imposing restrictions on the credentials that Users must own and the information from these credentials that Users must present in order to access the service. The Verifier's restrictions are described in its presentation policy. The User generates from her credentials a presentation token that contains the required information and the supporting cryptographic evidence. In the identity management literature, the Verifier is sometimes also referred to as the relying party, the server, or the service provider.
- A *Revocation Authority* is responsible for revoking issued credentials, so that these credentials can no longer be used to generate a presentation token. The use of a particular Revocation Authority may be imposed by the Issuer, in which case the revoked credentials cannot be used with any Verifier for any purpose, or by the Verifier, in which case the effect of the revocation is local to the Verifier and does not affect presentations with other Verifiers. Both the User

and the Verifier must obtain the most recent revocation information from the Revocation Authority to generate, respectively verify, presentation tokens.

- An *Inspector* is a trusted authority who can de-anonymize presentation tokens under specific circumstances. To make use of this feature, the Verifier must specify in the presentation policy which Inspector should be able to recover which attribute(s) under which circumstances. The User is therefore aware of the de-anonymization options when the token is generated and actively participates to make this possible; therefore the User can make a conscious decision based on her trust in the Inspector.

In an actual deployment, some of the above roles may actually be fulfilled by the same entity or split among many. For example, an Issuer can at the same time play the role of Revocation Authority and/or Inspector, or an Issuer could later also be the Verifier of tokens derived from credentials that it issued.

Moreover, some of the flows presented in this document could be adapted for particular deployments and scenarios. It is assumed that Verifiers already have in their possession or trust the Issuer and Revocation Authority data needed to validate a presentation token. Nothing prevents, however, a User to collect this data and present it to the verifier in a certified manner in a setup phase by piggybacking on an existing infrastructure (e.g., by signing the artifacts using an X.509 certificate). This would add flexibility to the system and allow dynamic trust establishments between the parties.

2.1 Credentials

A *credential* is a certified container of attributes issued by an Issuer to a User. An attribute is described by the *attribute type*, determining the semantics of the attribute (e.g., first name), and the *attribute value,* determining its contents (e.g., John). By issuing a credential, the Issuer vouches for the correctness of the contained attributes with respect to the User. The User can then later use her credentials to derive *presentation tokens* that reveal partial information about the encoded attributes to a Verifier.

The *credential specification* specifies the list of attribute types that are encoded in a credential. Since Privacy-ABCs natively only support integers of limited size (typically 256 bits) as attribute values, the credential specification also specifies how the attribute values are mapped to their integer representation. Depending on the data type and size of the attribute value, this encoding may involve a cryptographic hash to be applied.

At setup, the Issuer generates public *issuer parameters* and a secret *issuance key*. The issuer parameters are used by verifiers to verify the authenticity of presentation tokens. Trust management and distribution are out of scope of this specification; a standard public-key infrastructure (PKI), e.g., using hierarchical certification authorities, can be used to ensure that verifiers obtain authentic copies of the credential specifications and issuer parameters. Apart from cryptographic information, the issuer parameters also contain other meta-data such as the hash algorithm to use to create presentation tokens, as well as information for user binding, device binding, and revocation (see later). The Issuer keeps the issuance key strictly secret and uses it only to issue credentials.

2.2 Presentation

To provide certified information to a Verifier (for authentication or an access decision), the User uses one or more of her credentials to derive a *presentation token* and sends it to the Verifier. A single presentation token can contain information from any number of credentials. The token can reveal a subset of the attribute values in the credentials (e.g., IDcard.firstname = "John"), prove that a value satisfies a certain predicate (e.g., IDcard.birthdate < 1993/01/01) or that two values satisfy a predicate (e.g., IDcard.lastname = creditcard.lastname).

Apart from revealing information about credential attributes, the presentation token can optionally sign an application-specific message and/or a random nonce to guarantee freshness. Moreover, presentation tokens support a number of advanced features such as pseudonyms, device binding, inspection, and revocation that are described in more details below.

A Verifier announces in its *presentation policy* which credentials from which Issuers it accepts and which information the presentation token must reveal from these credentials. The Verifier can cryptographically verify the authenticity of a received presentation token using the credential specifications and issuer parameters of all credentials involved in the token. The Verifier must obtain the credential specifications and issuer parameters in a trusted manner, e.g., by using a traditional PKI to authenticate them or retrieving them from a trusted location.

The presentation token created in response to such a presentation policy consists of the *presentation token description*, containing a mechanism-agnostic description of the revealed information, and the *presentation token evidence,* containing opaque technology-specific cryptographic data in support of the token description.

Presentation tokens based on Privacy-ABCs are in principle cryptographically unlinkable and untraceable, meaning that Verifiers cannot tell whether two presentation tokens were derived from the same or from different credentials, and that Issuers cannot trace a presentation token back to the issuance of the underlying credentials. However, we will later discuss additional mechanisms that, with the User's consent, enable a dedicated third party to recover this link again (see Section 2.6 for more details).

Obviously, presentation tokens are only as unlinkable as the information they intentionally reveal. For example, tokens that explicitly reveal a unique attribute (e.g., the User’s social security number) are fully linkable. Moreover, pseudonyms and inspection can be used to purposely create linkability across presentation tokens (e.g., to maintain state across sessions by the same User) and create traceability of presentation tokens (e.g., for accountability reasons in case of abuse). Finally, Privacy-ABCs have to be combined with anonymous communication channels (e.g., Tor onion routing) to avoid linkability in the “layers below”, e.g., by the IP addresses in the underlying communication channels or by the physical characteristics of the hardware device on which the tokens were generated.

2.3 User Binding

Credentials can optionally be bound to a specific User by binding them to a *user secret* that is known only to that user. The user secret can be a cryptographically strong random value stored on a trusted device or cloud, or it could be a human-memorizable password or PIN code. The credential specification specifies whether the credentials issued according to this specification are to employ user binding or not.

A presentation token derived from a user-bound credential always contains an implicit proof of knowledge of the underlying user secret, so that the Verifier can be sure that the rightful owner of the credential was involved in the creation of the presentation token.

User binding can be used to prevent “credential pooling”, i.e., multiple Users sharing their credentials and user secrets in order to obtain access to services that they wouldn’t have access to individually. The issuance of a user-bound credential can optionally be performed in such a way that the newly issued credential is bound to the same user secret as an existing credential already owned by the User – without the Issuer learning the user secret in the process (see Section 2.7). A Verifier can also optionally impose in its presentation policy that all user-bound credentials involved in the creation of the token must be bound to the *same* user secret. Thereby, the user secret becomes a valuable “master secret” that, when revealed to a third party, allows the latter to take over the User’s entire digital identity.

As an extra protection layer, the credentials can also be bound to hardware devices (see Section 2.5) that are distributed in a restricted way so that each natural person can obtain at most a single device.

2.4 Pseudonyms

There are many situations where a controlled linkability of presentation tokens is actually desirable. For example, web services may want to maintain state information per user or user account to present a personalized interface, or conversation partners may want to be sure to continue a conversation thread with the same person that they started it with.

Privacy-ABCs have the concept of *pseudonyms* to obtain exactly such controlled linkability. If the user secret from Section 2.3 is seen as the equivalent of a User's secret key in a classical public-key authentication system, then a pseudonym is the equivalent of the User's public key. Just like a public key, it is derived from the user secret and can be given to a Verifier so that the User can later re-authenticate by proving knowledge of the user secret underlying the pseudonym. Unlike public keys of which there is only one for every secret key, however, Users can generate an unlimited number of unlinkable pseudonyms for a single user secret. Users can thus be known under different pseudonyms with different Verifiers, yet authenticate to all of them using the same user secret.

To be able to re-authenticate under a previously established pseudonym, the User may need to store some additional information used in the generation of the pseudonym. To assist the User in keeping track of which pseudonym she used at which Verifier, the Verifier's presentation policy specifies a *pseudonym scope*, which is just a string that the User can use as a key to look up the appropriate pseudonym. The scope string could for example be the identity of the Verifier or the URL of the web service that the User wants to access.

We distinguish between the following three types of pseudonyms:

- *Verifiable pseudonyms* are pseudonyms derived from an underlying user secret as described above, allowing the User to re-authenticate under the pseudonym by proving knowledge of the user secret. Presenting a verifiable pseudonym does not involve presenting a corresponding presentation token and is useful in login scenarios, e.g., to replace usernames/passwords.
- *Certified pseudonyms* are verifiable pseudonyms derived from a user secret that also underlies an issued credential. By imposing same-user binding in the presentation policy and token (see Section 2.3), a single presentation token can therefore prove ownership of a credential and at the same time establish a pseudonym based on the same user secret. As an example, a student could create several personas on a school discussion board using its core student credential, presenting the pseudonym associated with each persona, and without the possibility of anyone else (including a malicious Issuer) to present a matching pseudonym to hijack's the user's identity.
- *Scope-exclusive pseudonyms* are certified pseudonyms that are guaranteed to be unique per scope string and per user secret. For normal (i.e., non-scope-exclusive) pseudonyms, nothing prevents a User from generating multiple unlinkable pseudonyms for the same scope string. For scope-exclusive pseudonyms, it is cryptographically impossible to do so. By imposing a scope-exclusive pseudonym to be established, a Verifier can be sure that only a single pseudonym can be created for each credential or combination of credentials that are required in the presentation. This feature can be useful to implement a form of "consumption control" in situations (e.g., online petitions or one-time coupons) where users must remain anonymous to the Verifier but should not be allowed to create multiple identities based on a single credential. Note that scope-exclusive pseudonyms for different scope strings are still unlinkable, just like normal verifiable pseudonyms.

2.5 Device Binding

Similarly to user binding, a credential can also optionally be bound to a physical device such as a smart card. The trusted device contains a secret cryptographic key that never leaves the device, but the device does participate in the presentation token generation to include an implicit proof of knowledge of this key in the token. Presentation tokens of a device-bound credential always must contain such a proof, so that it is impossible to create a presentation token without the device.

The credential specification specifies whether credentials issued according to the specification are to employ device binding. Multiple credentials can be bound to the same device, and a presentation policy can optionally impose that all device-bound credentials used in the presentation token are bound to the same device. (By default, they could be bound to different devices. If this is the case, however, and same-device binding is not enforced, then the devices of all involved credentials must be used to create the presentation token.)

2.6 Inspection

Absolute user anonymity in online services can easily lead to abuses such as spam, harassment, or fraud. Privacy-ABCs give Verifiers the option to strike a trade-off between anonymity for honest users and accountability for misbehaving users through a feature called *inspection*.

An *Inspector* is a dedicated entity, separate from the Verifier, who can be asked to uncover one or more attributes of the User who created a presentation token, e.g., in case of abuse. The Inspector must on one hand be trusted by the User not to uncover identities unnecessarily, and must on the other hand be trusted by the Verifier to assist in the recovery when an abuse does occur.

Presentation tokens are fully anonymous by default, without possibility of inspection. To enable an Inspector to trace a presentation token when necessary, the presentation policy must explicitly specify the identity of the Inspector, which information the Inspector must be able to recover, and under which circumstances the Inspector can be asked to do so. The User then creates the presentation token in a particular way so that the Verifier can check by himself, i.e., without help from the Inspector, that the token could be inspected under the specified restrictions if necessary.

In more technical detail, the Inspector first sets up a public encryption key and a secret decryption key; he makes the former publicly available but keeps the latter secret. The presentation policy specifies

- (a reference to) the Inspector's public key,
- which attribute(s) from which credential(s) which Inspector must be able to recover, and
- the inspection grounds, i.e., an arbitrary human- and/or machine-readable string describing the circumstances under which the token can be inspected.

The User then prepares the presentation token so that it contains encrypted versions of the requested attribute values under the respective public key of the suggested Inspector, together with a verifiable cryptographic proof that the encryption contains the same attribute values as encoded in the User's credentials and certified by the Issuer.

When the situation described in the inspection grounds arises, the Inspection Requester can ask for an inspection. Besides the Verifier, other entities such as criminal prosecutors, courts or the User herself are also potential requesters for inspection. Usually the Verifier holds the stored copy of the presentation token and will send it to the Inspector for inspection, possibly together with some kind of evidence (e.g., transaction logs, inquiry of competent authority, court order) that the inspection grounds have been fulfilled. The inspection grounds are cryptographically tied to the presentation token, so the Verifier cannot change these after having received the token. The Inspector uses its secret key to decrypt the encrypted attribute values and returns the clear text values to the Inspection Requestor.

De-anonymization of presentation tokens is probably the main use case for inspection, but it can also be used to reveal useful attribute values to third parties instead of to the Verifier himself. For example, suppose the Verifier is an online merchant wishing to accept credit card payments without running the risk of having the stored credit card data stolen by hackers. In that case, he can make the User encrypt her credit card number under the public key of the bank by specifying the bank as an Inspector for the credit card number with “payment” as inspection grounds.

2.7 Credential Issuance

In the simplest setting, an Issuer issues credentials to Users “from scratch”, i.e., without relation to any existing credentials already owned by the Users. In this situation, the User typically has to convince the Issuer through some out-of-band mechanism that she qualifies for a credential with certain attribute values, e.g., by showing up in person at the Issuer’s office and showing a physical piece of ID, or by providing some bootstrap electronic identity. Credential issuance is a multi-round interactive protocol between the Issuer and the User. The attribute values can be specified by either parties, or jointly generated at random (i.e. the Issuer can be ensured an attribute value is chosen randomly and not chosen solely by User, but without the Issuer learning the attribute value).

Privacy-ABCs also support a more advanced form of credential issuance where the information embedded in the newly issued credential is “carried over” from existing credentials already owned by the User, without the Issuer being able to learn the carried-over information in the process. In particular, the newly issued credential can

3. carry over attribute values from an existing credential,
4. carry over “self-claimed” attribute values, i.e., values chosen by the User,
5. be bound to the same User (i.e., to the same user secret) as an existing credential or verifiable pseudonym (see Sections 2.3 and 2.4), and
6. be bound to the same device as an existing credential (see Section 2.5),

all without the Issuer being able to learn the carried-over attribute values, user secret, or device information in the process.

Moreover, the Issuer can insist that certain attributes be generated jointly at random, meaning that the attribute will be assigned a fresh random value. The Issuer does not learn the value of the attribute, but at the same time the User cannot choose, or even bias, the value assigned to the attribute. This feature is for instance helpful to impose usage limitation of a credential. To this end, the Issuer first embeds a jointly random value as serial number in the credential. A Verifier requesting a token based on such a credential can require that its serial number attribute must be disclosed by the User, such that it can detect if the same credential is used multiple times. The jointly random attribute hereby ensures that the Verifier and Issuer can not link the generated token and issued credential together, and the User can not cheat by biasing the serial number in the credential.

The Issuer publishes or sends to the User an *issuance policy* consisting of a presentation policy and a *credential template*. The presentation policy expresses which existing credentials the User must possess in order to be issued a new credential, using the same concepts and format as the presentation policy for normal token presentation (see Section 2.2). The User prepares a special presentation token that fulfills the stated presentation policy, but that contains additional cryptographic information to enable carrying over attribute, user binding, and device binding information. The credential template describes the relation of the new credential to the existing credentials used in the presentation token by specifying

- which attributes of the new credential will be assigned the same value as which attributes from which credential in the presentation token,
- whether the new credential will be bound to the same user secret as one of the credentials or pseudonyms in the presentation token, and if so, to which credential or pseudonym, and
- whether the new credential will be bound to the same device as one of the credentials in the presentation token, and if so, to which.

The User and Issuer subsequently engage in a multi-round issuance protocol, at the end of which the User obtains the requested credential.

2.8 Revocation

No identification system is complete without a proper means of revoking credentials. There can be many reasons to revoke a credential. For example, the credential and the related user or device secrets may have been compromised, the User may have lost her right to carry a credential, or some of her attribute values may have changed. Moreover, credentials may be revoked for a restricted set of purposes. For example, a football hooligan's digital identity card could be revoked for accessing sport stadiums, but is still valid for voting or submitting tax applications.

In classical public-key authentication systems, revocation usually works by letting either the Issuer or a dedicated Revocation Authority publish the serial numbers of revoked certificates in a so-called certificate revocation list. The Verifier then simply checks whether the serial number of a received certificate is on such a list or not. The same approach does not work for Privacy-ABCs, however, as Privacy-ABCs should not have a unique fingerprint value that must be revealed at each presentation, as this would nullify the unlinkability of the presentation tokens again. However, there are cryptographically more advanced revocation mechanisms that provide the same functionality in a privacy-preserving way, i.e., without imposing a unique trace on the presentation tokens. This document describes an abstract interface that covers all currently known revocation mechanisms.

Credentials are revoked by dedicated Revocation Authorities, which may be separate entities, or may also take the role of Issuer or Verifier. The Revocation Authority publishes its *revocation parameters* and regularly (e.g., at regular time intervals, or whenever a new credential is revoked) publishes the most recent *revocation information* that Verifiers use to make sure that the credentials used in a presentation token have not been revoked. The revocation parameters contain information where and how the Verifiers can obtain the most recent revocation information. Depending on the revocation mechanism, the identifiers of revoked credentials may or may not be visible from the revocation information. It is important that Verifiers obtain the most recent revocation information from the Revocation Authority directly, or that the revocation information is signed by the Revocation Authority if it is provided by the User together with the presentation token.

In order to prove that their credentials have not been revoked, Users may have to maintain *non-revocation evidence* for each credential and for each Revocation Authority against which the credential must be checked. The first time that a User checks one of her credentials against a particular Revocation Authority, she obtains an *initial non-revocation evidence*. Later, depending on the revocation mechanism used, the User may have to obtain regular *non-revocation evidence updates* at each update of the revocation information. Also depending on the revocation mechanism, these evidence updates may be the same for all Users/credentials or may be different for each User/credential. In the latter case, again depending on the mechanism, the Users may fetch their updates from a public bulletin board or obtain their updates over a secure channel.

We distinguish between two types of revocation. Apart from a small list of exceptions, all revocation mechanisms can be used for either type of revocation.

- In *Issuer-driven revocation*, the Issuer specifies as part of the issuer parameters the Revocation Authority (and revocation parameters) to be used when verifying a presentation token involving a credential issued by his issuer parameters. Issuer-driven revocation is always global in scope, meaning that any presentation token MUST always be checked against the most recent revocation information by the specified Revocation Authority, and that the Issuer denies any responsibility for revoked credentials. Issuer-driven revocation is typically used when credentials have been compromised or lost, or when the User is denied all further use of the credential. The Revocation Authority may be managed by or be the same entity as the Issuer, or may be a separate entity. Issuer-driven revocation is performed through a *revocation handle*, a dedicated unique identifier that the Issuer embeds as an attribute in each issued credential (but which by default should not be revealed in a presentation token). When the Issuer, a Verifier, or any third party wants to revoke a credential, it must provide the revocation handle to the Revocation Authority. How the party requesting the revocation learns the revocation handle is out of the scope of this document; this could for example be done digitally by insisting in the presentation policy that the revocation handle be revealed to a trusted Inspector, or physically by arresting the person and obtaining his or her identity card.
- In *Verifier-driven revocation*, the Verifier specifies as part of the presentation policy against which Revocation Authority or Authorities (and revocation parameters) the presentation must *additionally* be checked, i.e., on top of any Revocation Authorities specified by the Issuer in the issuer parameters. The effect of the revocation is local to those Verifiers who explicitly specify the Revocation Authority in their presentation policies, and does not affect presentations with other Verifiers. Verifier-driven revocation is mainly useful for purpose-specific revocation (e.g., a no-fly list for terrorists) or verifier-local revocation (e.g., a website excluding misbehaving users from its site). Note that if unlinkability of presentation tokens is not a requirement, the latter effect can also be obtained by using scope-exclusive pseudonyms. The Revocation Authority may be managed by or be the same entity as the Verifier, or may be a separate entity. Verifier-driven revocation can be performed based on any attribute, not just based on the revocation handle as for Issuer-driven revocation. It is up to the Verifier and/or the Revocation Authority to choose an attribute that on the one hand is sufficiently identifying to avoid false positives (e.g., the User's first name probably doesn't suffice) and on the other hand will be known to the party likely to request the revocation of a credential. Verifier-driven revocation is essentially a black list of attribute values, banning all credentials with a blacklisted attribute value.

3 Architecture

This chapter describes the architecture of (Privacy-)ABC systems, their components and the relations among those. Following standard design principles, our architecture uses a layered approach, where related functionalities are grouped into a common layer that provides simple interfaces towards other layers and components, thereby abstracting the internal design and structure. As mentioned in Chapter 1, the architecture focuses on the technology-independent components for (Privacy-)ABC-systems, grouped in the ABCE-layer, which can be integrated in various application and deployment scenarios. That is, we do not propose a concrete application-level deployment but provide generic interfaces to the ABCE layer that allow for a flexible integration. Note that we aim at an architecture that is capable of supporting all the privacy-enhancing features of privacy-ABC, but at the same time is not exclusive to those, i.e, it is also generic enough to support "standard" ABC technologies such as X.509 certificates.

We start by describing the main functionalities of the different layers and components in Section 3.1. We then describe the internal components and dependencies of the ABCE-layer along the main use-cases. That is, in Section 3.2 we provide an overview of the setup-functionalities that are provided by the ABCE-layer. Section 3.3 is devoted to the presentation of tokens, thereby describing the steps that a User and Verifier have to perform in order to create and to verify a presentation token. The process of the issuance of a credential is described in Section 3.4, and can incorporate some of the presentation steps described in the previous section. Section 3.5 then deals with the inspection process that can be used to reveal some previously hidden attributes, and Section 3.6 describes the ABCE functionalities in the context of revocation.

This chapter assumes that the reader is already familiar with the general features and concepts of Privacy-ABCs (see Chapter 2 otherwise) and gives a high-level description of the Privacy-ABC-core architecture and its components. Thus, it can also be seen as an introduction to Chapter 4 which describes the data formats that are exchanged among Privacy-ABC entities and to Chapter 5 that presents the application programming interfaces (API) of the ABCE layer components. Note that this chapter already refers to the *external* methods provided by the ABCE layer which are described in more detail in Section 5.1.

3.1 Architectural Design

The Privacy-ABC architecture defines for each entity the core-components required to operate with attribute-based credentials. As an example, Figure 3.1 shows an overview of the components for the User's side.

For the sake of completeness, the figure also shows an application layer. As mentioned before, this layer is not part of the Privacy-ABC architecture, but will operate on top of that. Roughly, this layer comprises all application-level components, which in the case of the User-side deployment includes the main application and the *IdentitySelection* (see description below). The application layer of Verifiers and Issuers will also contain the policy store and access control engine.

The ABCE *layer* contains all technology-agnostic methods and components for a (Privacy-)ABC system. That is, it contains e.g. the methods to parse an obtained presentation policy, perform the selection of applicable credentials for a given policy or to trigger the mechanism-specific generation or verification of the cryptographic evidence. The ABCE-layer is invoked by the application-layer and calls out the *CryptoEngine* to obtain the mechanism-specific cryptographic data. To perform their tasks, the internal components can also make use of other external components such as the *KeyManager*, *RevocationProxy* or *IdentitySelection*. The ABCE-internal components are described in Section 3.2-3.6 along the different use-cases.

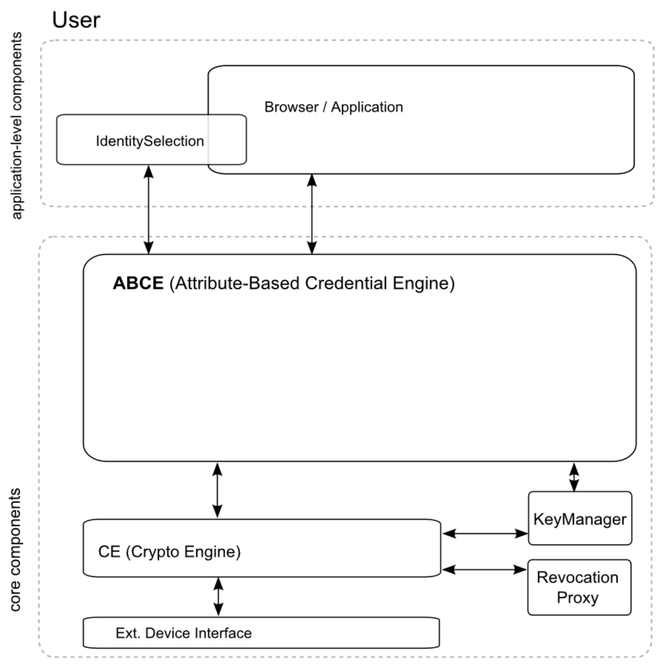

Figure 3.1 - Overview of the ABC Architecture on the User Side

The *KeyManager* deals with the (cryptographic) keys of all parties and keeps them up to date (key life cycle management). On input of an identifier (URI) for a key, it returns a (list of) cryptographic key(s) that are currently valid for that URI. This component takes also care of fetching the current (public) revocation information, that will be needed to keep the credentials up to date, or to verify whether a received presentation token is still valid.

The *CryptoEngine* provides common interfaces to generate the cryptographic information required e.g., to create, present, verify or inspect a presentation/issuance token. It internally orchestrates and performs the mechanism-specific cryptographic methods, such as the computation of signatures (e.g., Idemix, U-Prove signature), commitments, zero-knowledge proofs, etc.

The *RevocationProxy* handles the communication between the *CryptoEngine* and the Revocation Authority for the generation or presentation of tokens/credentials that are subject to revocation. The concrete communication pattern strongly depends on the specific revocation mechanisms.

The *IdentitySelection* component provides methods, possibly presented by a graphical user interface, to support a User in choosing a preferred combination of credential and/or pseudonyms, if there are different possibilities to satisfy a given presentation policy. A user interface is also used to obtain User consent, whenever personal data is revealed.

The *DeviceInterface* components provide optional generic interfaces to ease the integration of external devices, such as smart cards, for both the "outsourcing" of computation and also to obtain data stored externally on the device. The integration of an external device might for instance be necessary if device binding of a token is required (see Chapter 2 for more details).

3.2 Setup

In order to furnish all parties in a Privacy-ABC setting with the required key-material, the ABCE-layer provides for each party a dedicated method to obtain its private and public (if any) cryptographic parameters. Private keys will be stored in the trusted storage of the corresponding party.

In particular, the ABCE-layer provides the following setup methods (see Chapter 5 for the detailed specification)

- `setupUser()` generates a cryptographically strong user secret.
- `setupSystemParameters()` generates system wide public parameters for Issuers, e.g., this method can be invoked by an Issuer itself or an independent entity if several Issuers share the the same system parameters.
- `setupIssuerParameter()` generates a secret issuance key and public Issuer parameters for the given system parameters and a given credential specification.
- `setupRevocationAuthorityParameter()` generates a secret Revocation Authority key, public Revocation Authority parameters and the initial revocation information.
- `setupInspectionPublicKey()` generates a secret decryption key and corresponding public encryption key an Inspector.

3.3 Presentation of a Token

The process of the presentation of a token is triggered when the application on the User's side contacts a Verifier to request access to a resource (Figure 3.2 – step 1). Having received the request, the Verifier responds with one or more Presentation Policies. A Presentation Policy defines what data a user has to reveal to the Verifier in order to gain access to the requested resource. That is, it defines e.g. which credentials from which trusted issuers are required, which attributes from those credentials have to be revealed, or which conditions the attributes have to fulfil. If there are several alternatives of applicable policies the server responds with a set of presentation policies. A detailed specification of a presentation policy is given in Chapter 4. Upon receiving the policy, the application of the user's side invokes the ABCE layer by calling the `createToken()` method with the received presentation policy (Figure 3.2 – step 2.b).

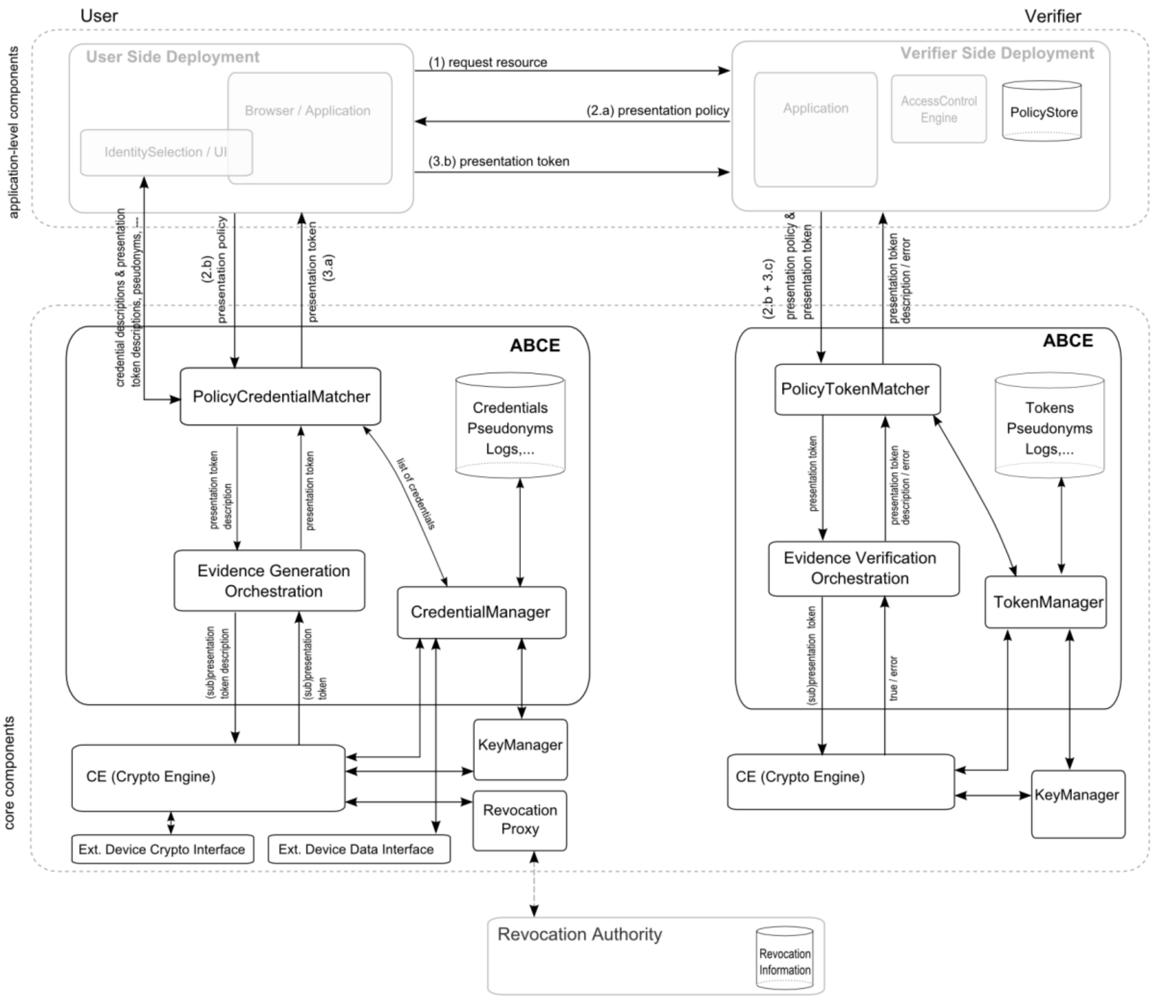
User
Verifier
application-level components
User Side Deployment
Browser / Application
IdentitySelection / UI
(1) request resource
(2.a) presentation policy
(3.b) presentation token
Verifier Side Deployment
Application
AccessControl Engine
PolicyStore
credential descriptions & presentation token descriptions, pseudonyms, ...
(2.b) presentation policy
presentation token (3.a)
(2.b + 3.c) presentation policy & presentation token
presentation token description / error
ABCE
PolicyCredentialMatcher
Credentials Pseudonyms Logs,...
presentation token description
presentation token
list of credentials
Evidence Generation Orchestration
CredentialManager
(sub)presentation token description
(sub)presentation token
PolicyTokenMatcher
Tokens Pseudonyms Logs,...
presentation token
presentation token description / error
Evidence Verification Orchestration
TokenManager
(sub)presentation token
true / error
core components
CE (Crypto Engine)
KeyManager
Revocation Proxy
Ext. Device Crypto Interface
Ext. Device Data Interface
Revocation Authority
Revocation Information

Figure 3.2 - Presentation of a Token

Inside the ABCE layer, this call is forwarded to the *PolicyCredentialMatcher*, which upon receiving the presentation policies as an input, communicates with the *CredentialManager* to generate a list of possible credentials and/or established pseudonyms, together with the corresponding presentation token descriptions which each will satisfy one of the presentation policies. It then invokes the *IdentitySelection/UI* to get user consent to the generated presentation token description, or in case there are several possibilities to satisfy a policy, to select the preferred presentation token description first. In the latter case, it also passes the selection and the presentation policies to the *IdentitySelection/UI* component such that the user or an automated process can choose a presentation token description and subset of the credentials that shall be used to generate the presentation token. This choice is passed back to the *PolicyCredentialMatcher*, which subsequently invokes the *CredentialManager* to bring all required credentials which are subject to revocation up to date with respect to the current revocation information. For this operation it will rely on the *KeyManager* and the *CryptoEngine*. If all credentials are updated (if necessary), the *PolicyCredentialMatcher* triggers the *EvidenceGenerationOrchestration* to generate the presentation token for the chosen presentation token description and identifiers of the selected credentials. As a presentation token may consist of different token formats, e.g., X.509 certificates and Idemix/U-Prove tokens, the *EvidenceGenerationOrchestration* takes care of first generating these sub-tokens using the *CryptoEngine* and then assembling them into a single presentation token. To this end, it dissects the presentation token description and invokes the *CryptoEngine* for each generated sub-presentation token description and the corresponding credential identifier(s).

The *CryptoEngine* on input a (sub-)presentation token description and a set of credential identifiers, creates the cryptographic evidence that supports the token description. To do so, the *CryptoEngine* can use the *CredentialManager* to retrieve the required credentials, or pseudonym data from the credential store and the *KeyManager* to obtain the required public keys. The *CryptoEngine* can also call out to an external device if requested by the token description, e.g., if device-binding is required. It finally returns the cryptographic (sub-)evidence to the *EvidenceGenerationOrchestration*. When the *EvidenceGenerationOrchestration* received the evidence for all (sub-) presentation token descriptions, it merges them into the final crypto evidence and returns the full presentation token through the *PolicyCredentialMatcher* to the application layer (Figure 3.2 – step 3.a).

The User's application then sends the presentation token to the Verifier (Figure 3.2 – step 3.b). The Verifier's application passes the received presentation token and the previously sent presentation policy to its ABCE layer (Figure 3.2 – step 2.b+3.c), by calling the `verifyTokenAgainstPolicy()` method. Inside the ABCE layer this call gets forwarded to the *PolicyTokenMatcher,* which verifies in two steps whether the presentation token satisfies the presentation policy. First, it checks whether the statements made in the presentation token description satisfy the required statements in the presentation policy. If the policy requested the re-use of an established pseudonym, the *PolicyTokenMatcher* calls on the *TokenManager* to look up if a presented pseudonym already exists. When the first check succeeds, i.e., the presentation token description matches the presentation policy, it subsequently invokes the *EvidenceVerificationOrchestration* with the presentation token, which then verifies the validity of the crypto evidence. To this end, the *EvidenceVerificationOrchestration* calls the *CryptoEngine* for each sub-evidence with corresponding sub-presentation token. If the combined evidence supports the token description, it returns "valid" to the *PolicyTokenMatcher*. In case that both checks `succeed`, the *PolicyTokenMatcher* stores the token in a dedicated store (if requested by the application) and returns a description of the token to the application layer. This description includes a unique identifier, which allows the application to retrieve the token later from the store. If one of the checks fails, a list of error messages is returned to the application.

3.4 Issuance of a Credential

Roughly, issuance of a credential is an interactive protocol between a User and an Issuer, where at the end the User obtains a credential, i.e., a certified list of attribute-value pairs (or an error message, in case the protocol failed). In fact, issuance can be seen as a special case of a normal resource request, where the resource is a new credential the user wants to obtain. Thus, to handle such a credential request the ABCE layer might invoke the same components and procedures as in the presentation scenario described above. However, the issuance scenario also requires an additional ABCE component and additional or modified steps on the other components to allow e.g., for "carried-over" attributes. That is, attributes can be “carried over” from existing credentials already owned by the User into the newly issued one, in a way such that the Issuer does not learn those values.

To start an issuance process the User first authenticates toward the Issuer (Figure 3.3 – step 1). How the authentication is done, is outside of the scope of the Privacy-ABC architecture, i.e., it can even be done using non-ABC technologies such as standard username-password checks. After, or together, with the authentication, the User sends a credential request which specifies the credential type he wants to obtain (Figure 3.3 - step 2). As described in Section 2.7, the subsequent issuance protocol can come in different ways.

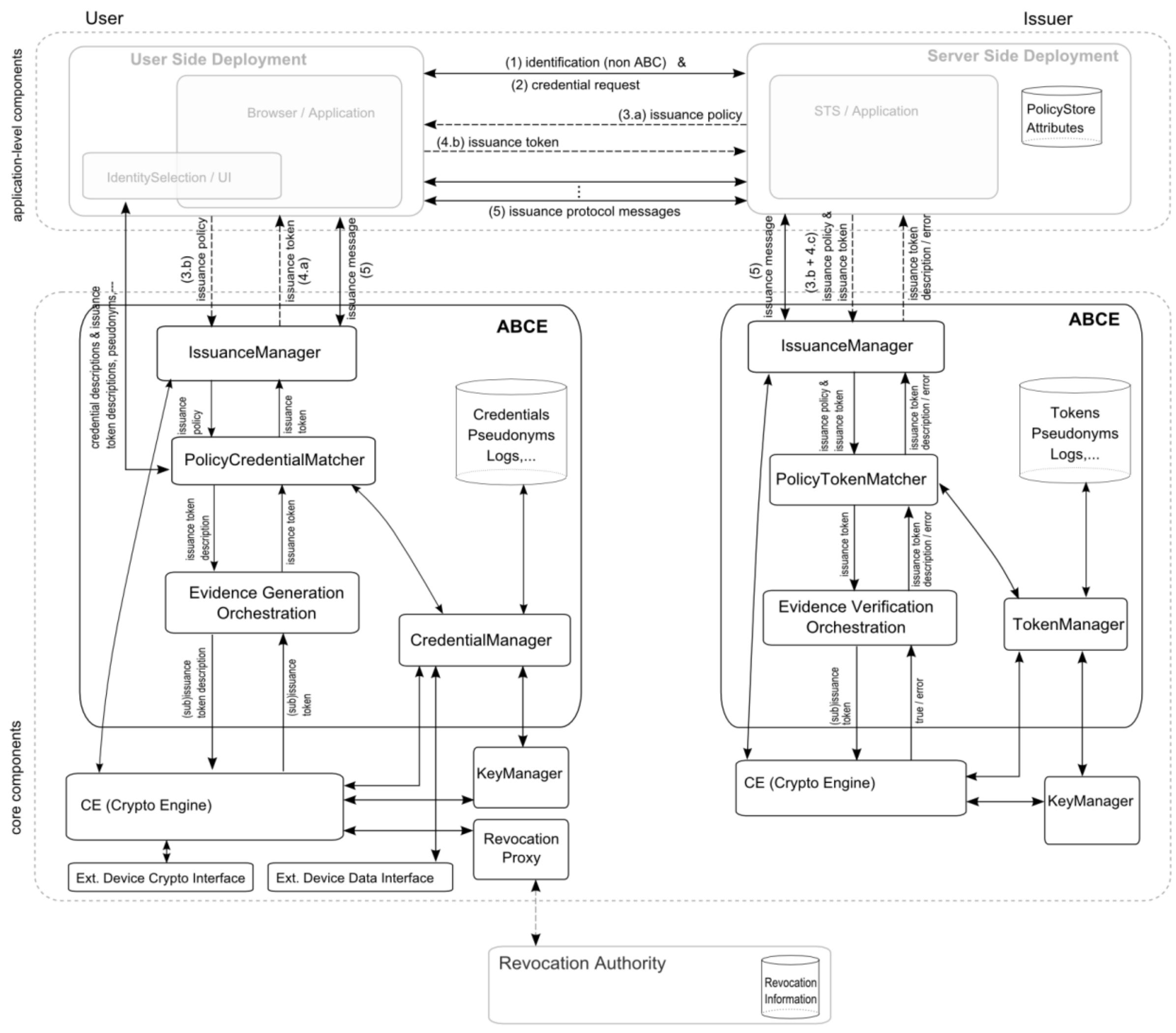

Figure 3.3 - Issuance of a Credential

3.4.1 Issuance "from scratch"

In the most simple setting an Issuer issues credentials to Users “from scratch”, i.e., without relation to any existing credentials already owned by the Users. Thus, in such a setting, the Issuer will start directly the issuance protocol (Figure 3.3 – step 5) by invoking the ABCE layer on the `initIssuanceProtocol()` method. This method gets as input the Issuer certified attributes and an "empty" issuance policy, i.e., an issuance policy that contains solely the requested credential specification identifier, but not a credential template. Note that even in this simple setting, credential issuance can be a multi-round interactive protocol between the Issuer and the User. Such an issuance protocol is orchestrated on both sides by the *IssuanceManager* component, which in the simple setting will directly invoke the *CryptoEngine* of the User and Issuer to obtain the issuance messages. To link the possibly several issuance message together (in case a User or Issuer runs several issuance sessions in parallel), every message must contain a context attribute which must be the same for all messages belonging to one issuance protocol. If the requested credential is subject to Issuer-driven revocation, the *CryptoEngine* also takes care of obtaining the required revocation information and non-revocation evidence, for which it may use the *RevocationProxy* which will contact the *RevocationAuthority* specified in the credential specification.

Upon receiving a new issuance message, both the User and Issuer pass that message to their ABCE layer using the *issuanceProtocolStep()* method. If the output of that method is another issuance message this is sent back to the other party. At the end of a successful issuance protocol, the User stores his freshly generated credential in its local credential store.

3.4.2 Issuance with advanced features

In a more advanced setting, the information embedded in the newly issued credential can be invisibly "carried over" from existing credentials already owned by the User. To this end, the issuance protocol is preceded by the generation and verification of an issuance token. Thus, the Issuer first sends an issuance policy (Figure 3.3 – step 3.a), which consists of a presentation policy and a credential template. The presentation policy expresses which existing credentials the User must possess in order to be issued a new credential, using the same concepts and format as the presentation policy for normal token presentation. The credential template describes the relation of the new credential to the existing credentials used in the presentation token by specifying e.g. which attribute values from the credentials in the presentation token will be reused in the new credential, and where. The User then passes the issuance policy and possibly a set of self-claimed attributes that were already specified/suggested by the application layer to its ABCE layer by calling the `createIssuanceToken()` method (Figure 3.3 – step 3.b).

The ABCE-layer passes this call to the *IssuanceManager*, which prepares an issuance token, i.e., a special presentation token that fulfils the stated issuance policy, but that also contains additional information to enable carried-over attributes, user binding and device binding. The *IssuanceManager* first invokes the *PolicyCredentialMatcher* with the received issuance policy. As in the presentation scenario, the *PolicyCredentialMatcher* then investigates whether the User has the necessary credentials and/or established pseudonyms to create an issuance token that satisfies the issuance policy. If there are multiple ways in which the policy can be satisfied (e.g., by using different sets of credentials), this method will invoke the *IdentitySelection/UI* to determine the preferred way of generating the issuance token, and also to extend or modify the set of self-claimed attributes which will be provided by the User to the credential. The *IdentitySelection/UI* is also used to get User consent to the revealed attributes in the issuance token. As in the presentation scenario, the *PolicyCredentialMatcher* subsequently calls on the *EvidenceGenerationOrchestration* to obtain the cryptographic evidence for the token. Here the evidence provided by the *CryptoEngine* can also contain additional cryptographic data which will subsequently be used in the issuance protocol, e.g., for the carried-over attributes. As in the simple issuance setting, the *CryptoEngine* further takes care of generating the required data to enable issuer-driven revocation, if requested by the credential specification. The issuance token itself is then wrapped in an issuance message and gets passed back to the Issuer through the application layer of the User (Figure 3.3 – step 4.a+4.b).

On the Issuer's side, receiving an issuance token triggers the `initIssuanceProtocol()` method of the ABCE layer with input the issuance policy (Figure 3.3 – step 3.b), an issuance message containing the issuance token (Figure 3.3 – step 4.c) and attributes certified by the Issuer. This call is passed to the *IssuanceManager*, which first verifies the issuance token with the help of the *PolicyTokenMatcher* and the *EvidenceVerificationOrchestration* component. If both checks succeed, i.e., the token description satisfies the issuance policy and the cryptographic evidence supports the token description, the *IssuanceManager* invokes the *CryptoEngine* to obtain the first message of the issuance protocol, which is passed via the application layer to the User.

In case of a multi-round issuance protocol, the User and Issuer exchange further issuance messages, obtained from calling on their `issuanceProtocolStep()` methods provided by the ABCE layer (Figure 3.3 – step 5). At the end of the protocol the User obtains the requested credential according to the credential template.

3.5 Inspection

As described in detail in Section 2.6, the inspection of credentials allows lifting the full anonymity that is usually provided by Privacy-ABC based presentation tokens in case of abuse or misbehaving Users. If inspectable attributes are requested by a Verifier, the presentation token of the User are specially prepared, such that the attributes in question are not revealed to the Verifier, but are verifiably encrypted in the token (under the public key of the Inspector) and tied to some inspection ground that were accepted by the User. In case the event specified in the inspection grounds occurred, the Inspection Requestor (e.g., the Verifier) can contact the Inspector and request the "de-anonymization" of a presentation token. In case inspection is triggered by the Verifier, he first fetches the presentation token using the `getToken()` method of the ABCE-layer. It then sends the received presentation token with the (non-cryptographic) evidence that the inspection grounds are fulfilled to the Inspector. If the Inspector accepts the evidence (the verification whether the evidence meets the inspection grounds is out of the scope of the Privacy-ABC-system), he invokes the ABCE-layer on the `Inspect()` method, which is directly forwarded to the *CryptoEngine*. Therein, the inspectable attributes are decrypted and returned to the application layer of the Inspector.

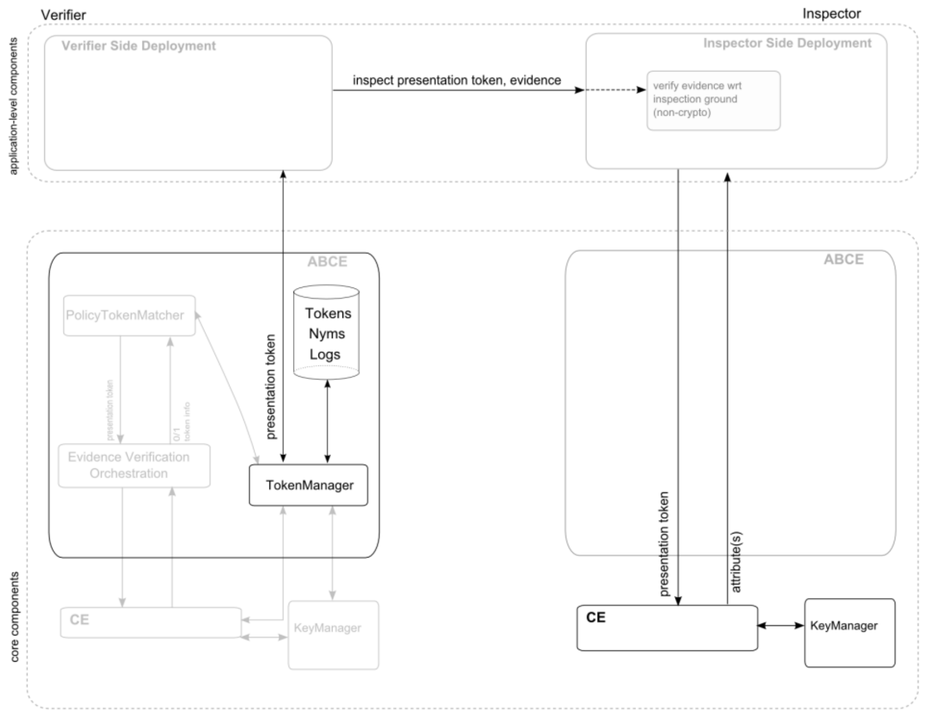

Figure 3.4 - Inspection

3.6 Revocation

The ABCE layer provides several interfaces to support revocation. For instance, Users and Verifiers can obtain recent revocation information by contacting the Revocation Authority which will then invoke its `getCurrentRevocationInformation()` for a given identifier of Revocation Authority parameters. If revocation requires an initialisation of the User with the Revocation Authority this will be detected and handled by the *CryptoEngine* of the User, that can invoke the *RevocationProxy* which in turn contacts the Revocation Authority where the `processRevocationMessage()` method will be triggered. This method can also be used to obtain user-specific updates of the private non-revocation evidence.

In case a credential should be revoked (i.e., Issuer-driven revocation) or a (set of) attributes should be "black-listed" by a Verifier/3rd Party (i.e., Verifier-driven revocation) the corresponding Revocation Authority calls the `revoke()` method of the ABCE-layer, on input the given attribute(s). For Issuer-driven revocation this attribute will be the unique revocation handle of the credential that is supposed to be revoked.

4 ABC4Trust Protocol Specification

Given the multitude of distributed entities involved in a full-fledged Privacy-ABC system, the communication formats using which these entities interact must be fixed. Rather than profiling an existing standard format for identity management protocols such as SAML, WS-Trust, or OpenID, we felt that the many unique features of Privacy-ABCs were more suitably addressed by defining a dedicated format. In particular, existing standards do not support typical Privacy-ABC features such as pseudonyms, inspection, privacy-enhanced revocation, or advanced issuance protocols. In Chapter 8, we discuss how our Privacy-ABC infrastructure could be integrated with a number of existing frameworks.

This chapter provides the specification for data artifacts exchanged during the issuance, presentation, revocation, and inspection of privacy-enhancing attribute-based credentials for use in the ABC4Trust project. Our specification separates the mechanism-independent information conveyed by the artifacts from the opaque mechanism-specific cryptographic data. This specification only defines the format for the mechanism-independent information. It provides anchor points for where instantiating technologies, in particular, U-Prove and Identity Mixer, can insert mechanism-specific data, but does not fix standard formats for this data.

For the specification we use XML notation in the spirit of XML Schema, but refrain from providing a full-fledged XML Schema specification within this document for the sake of readability; we will, however, make available a separate XML schema file for the artifacts defined here. Although the artifacts are defined in XML, one could create a profile using a different encoding (ASN.1, JSON, etc.) See the corresponding schema file for more details.

We start in Section 4.1 with introducing the terminology and notation used throughout this chapter. Section 4.2 then provides the artifacts for the setup of the different Privacy-ABC entities, which includes e.g., the description of the credential type and the public parameters of an Issuer and Inspector. In Section 4.3 the specifications for all artifacts related to revocation are given. Section 4.4 is then dedicated to the Issuance of a credential and provides artifacts for the issuance policy and issuance token. For the presentation of a token, the corresponding specifications of a presentation policy and a presentation token are introduced in Section 4.4.

4.1 Terminology and Notation

4.1.1 Notational Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, and “MAY” in this document are to be interpreted as described in [RFC2119].

This specification uses the following syntax to define outlines for XML data:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.

- Characters are appended to elements and attributes to indicate cardinality:

  “?” (0 or 1)

  “*” (0 or more)

  “+” (1 or more)

- The character “|” is used to indicate a choice between elements.
- The characters “(“ and “)” are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- XML namespace prefixes (see Table) are used to indicate the namespace of the element being defined.
- XML elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions.

4.1.2 Namespaces

The base XML namespace URI used by the definitions in this document is as follows:

Prefix	XML namespace	Specification
xs	http://www.w3.org/2001/XMLSchema	[XMLSchema2]
abc	http://abc4trust.eu/wp2	This document

Table 4.1 - XML namespaces

4.2 Setup

4.2.1 Credential Specification

The credential specification describes the contents of the credentials. It can be created by the issuer, or by any external authority so that multiple issuers can issue credentials of the same specification. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a XML-signed document or provided as part of some metadata retrievable from a trusted source.

```
<abc:CredentialSpecification Version=”1.0” UserBinding=”xs:boolean”
DeviceBinding=”xs:boolean” Revocable="xs:boolean">
<abc:SpecificationUID>xs:anyURI</abc:SpecificationUID>
  <abc:AttributeDescriptions MaxLength=”xs:unsignedInt”>
    <abc:AttributeDescription Type=”xs:anyURI”
     DataType=”xs:anyURI” Encoding=”xs:anyURI”/>*
  </abc:AttributeDescriptions>
</abc:CredentialSpecification>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:CredentialSpecification`

This element contains the credential specification defining the contents of issued credentials adhering to this specification.

`/abc:CredentialSpecification/@Version`

This attribute indicates the version of this specification. The value MUST be “1.0”.

`/abc:CredentialSpecification/@UserBinding`

This attribute indicates whether credentials adhering to this specification must be bound to a user through an underlying user secret. See Section 2.3 for more information on user binding.

`/abc:CredentialSpecification/@DeviceBinding`

This attribute indicates whether credentials adhering to this specification must be bound to a physical device. See Section 2.5 for more information on device binding.

`/abc:CredentialSpecification/@Revocable`

This attribute indicates whether credentials adhering to this specification are revocable or not. If the `Revocable` attribute is set to true, then this credential specification MUST contain a dedicated attribute with attribute type `http://abc4trust.eu/wp2/abcschemav1.0/revocationhandle` for the revocation handle.

It is up to the Issuer to ensure that the value of the revocation handle is unique for each issued credential, so that credentials can be revoked individually based on its value. Even though there are no syntactical restrictions imposing this, presentation policies SHOULD NOT request to reveal the value of the revocation handle, as doing so enables Verifiers to link presentations tokens generated with the same credential. If necessary, inspection can be used to only reveal the value of the revocation handle under specific circumstances.

`/abc:CredentialSpecification/abc:SpecificationUID`

This element contains a URI that uniquely identifies the credential specification.

`/abc:CredentialSpecification/abc:AttributeDescriptions`

This element contains the descriptions of the attributes issued using this specification, encoded in order in the *n* child elements. It is empty if *n*=0, i.e., if `abc:AttributeDescriptions` has no child elements.

`…/abc:AttributeDescriptions/abc:AttributeDescription`

This element contains the description of one credential attribute.

`…/abc:AttributeDescriptions/abc:AttributeDescription/@MaxLength`

This attribute specifies the maximal length in bits of the integers to which attribute values are mapped using the encoding function. The keylength of any Issuer Parameters used to issue credentials adhering to this credential specification must be large enough so that attributes of the bitlength specified here can be supported. It is up to each specific credential mechanism to describe which keylenght supports which attribute bitlength.

`…/abc:AttributeDescriptions/abc:AttributeDescription/@Type`

This attribute contains the unique identifier of an attribute type encoded in credentials adhering to this specification. The attribute type is a URI, to which a semantics is associated by the definition of the attribute type. The definition of attribute types is outside the scope of this document; we refer to Section 7.5 in [IMI1.0] for examples. The attribute type (e.g., `http://example.com/firstname`) is *not* to be confused with the data type (e.g., `xs:string`) that is specified by the `DataType` attribute.

`…/abc:AttributeDescriptions/abc:AttributeDescription/@DataType`

This attribute contains the data type of the credential attribute. There is no restriction on which data types can be used, but usage of the XML Schema built-in data types is encouraged, as these are the only ones that can be used with attribute predicates.

`…/abc:AttributeDescriptions/abc:AttributeDescription/@Encoding`

To be embedded in a Privacy-ABC, credential attribute values must typically be mapped to 256-bit integers. This XML attribute specifies how the value of this credential attribute is mapped to an integer. Depending on the data type of the credential attribute, the encoding algorithm may specify a cryptographic hash function to be applied to map long values to an integer of the appropriate range. To support proving predicates comparing the values of different credential attributes (see Section 4.5.3), the involved attributes MUST use the same encoding.

4.2.2 Issuer Parameters

In order to issue credentials, the issuer must specify system parameters, and generate a key pair consisting of a secret issuing key and a public verification key. The issuer publishes its public parameters using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a certificate signed by a certification authority, or could be provided as part of some metadata retrievable from a trusted source.

```
<abc:IssuerParameters Version=”1.0”>
  <abc:ParametersUID>xs:anyURI</abc:ParametersUID>
  <abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
  <abc:SystemParameters>…</abc:SystemParameters>
  <abc:CredentialSpecUID>xs:anyURI</abc:CredentialSpecUID>
  <abc:HashAlgorithm>xs:anyUID</abc:HashAlgorithm>
  <abc:CryptoParams>…</abc:CryptoParams>
  <abc:UserBindingInfo>…</abc:UserBindingInfo>?
<abc:DeviceBindingInfo>…</abc:DeviceBindingInfo>?
<abc:RevocationParametersUID>…</abc:RevocationParametersUID>?
</abc:IssuerParameters>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuerParameters`

This element contains an issuer's public parameters.

`/abc:IssuerParameters/@Version`

This attribute indicates the version of this specification. The value MUST be “1.0”.

`/abc:IssuerParameters/abc:ParametersUID`

This element contains a URI that uniquely identifies the public issuer parameters.

`/abc:IssuerParameters/abc:AlgorithmID`

This element identifies the algorithm of the public issuer parameters (e.g., U-Prove, Identity Mixer).

`/abc:IssuerParameters/abc:SystemParameters`

This element contains the cryptographic system parameters that can be shared among many issuers. The `AlgorithmID` element determines how to parse this element.

`/abc:IssuerParameters/abc:CredentialSpecUID`

This element contains a URI that uniquely identifies the credential type that is issued by the issuer.

`/abc:IssuerParameters/abc:HashAlgorithm`

This element specifies the hash algorithm that is to be used in the generation of the presentation tokens derived from credentials issued under these parameters. This hash algorithm is not to be confused with the encoding algorithm that maps attribute values to integers and may also specify a hash function to apply to long attribute values.

`/abc:IssuerParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to issue, use, and verify credentials. The content of this element is defined in an external profile based on the value of the abc:AlgorithmID element.

`/abc:IssuerParameters/abc:UserBindingInfo`

This optional element contains additional cryptographic information for when these Issuer Parameters are used to issue credentials with user binding. The content of this element is technology-specific.

`/abc:IssuerParameters/abc:DeviceBindingInfo`

This optional element contains additional cryptographic information for when these Issuer Parameters are used to issue credentials with device binding. The content of this element is technology-specific.

`/abc:IssuerParameters/abc:RevocationAuthorityParametersUID`

This optional element contains the parameters identifier of a revocation authority that is responsible for revoking credentials issued under these issuer parameters. The parameters referred to here are determined by the issuer (i.e., issuer-driven revocation), meaning that any presentation token involving credentials issued under these issuer parameters MUST be checked against the latest revocation information associated to the revocation parameters referenced by this element.

4.2.3 Inspector Public Key

In order to decrypt encrypted attributes, an inspector must generate a key pair consisting of a secret decryption key and a public encryption key. The inspector publishes its public key using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a certificate signed by a certification authority, or could be provided as part of some metadata retrievable from a trusted source.

```
<abc:InspectorPublicKey Version=”1.0”>
  <abc:PublicKeyUID>xs:anyURI</abc:PublicKeyUID>
  <abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
  <abc:CryptoParams>…</abc:CryptoParams>
</abc:InspectorPublicKey>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:InspectorPublicKey`

This element contains an inspector's public key.

`/abc:InspectorPublicKey/@Version`

This attribute indicates the version of this specification. The value MUST be “1.0”.

`/abc:InspectorPublicKey/abc:PublicKeyUID`

This element contains a URI that uniquely identifies the public key.

`/abc:InspectorPublicKey/abc:AlgorithmID`

This element identifies the algorithm of the public key.

`/abc:InspectorPublicKey/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to issue, use, and verify credentials. The content of this element is defined in an external profile based on the value of the `abc:AlgorithmID` element.

4.3 Revocation

A Revocation Authority maintains information about valid and, in particular, revoked credentials. To do so, it first generates public parameters and possibly corresponding secret parameters. It publishes its public parameters together with a description of the particular revocation method that is used and a reference to the location where the most current revocation information will be published.

Some revocation mechanisms require users to obtain an additional piece of information called non-revocation evidence in order to be able to prove that their credential is still valid.

The different revocation mechanisms vary quite strongly in how the non-revocation is created and maintained. Depending on the specific mechanism, the non-revocation evidence

- may be the same for all users, or may be different for each user and/or each issued credential;
- may be sensitive information that the user needs to keep strictly secret, or may be leaked to other participants without further harm;
- may be first created during the issuance of the credential, during the first usage (presentation) of the credential, or at any time between issuance and first usage;
- may have to be kept up-to-date with the non-revocation information, or may remain the same for the lifetime of the credential.

The Revocation Authority can also include references to the locations where the users can obtain the information to create and to update their non-revocation evidence. Both the initialization of the non-revocation evidence and the update may be multi-leg cryptographic protocols.

4.3.1 Revocation Authority Parameters

Each Revocation Authority generates and publishes its parameters at setup. The parameters are static, i.e., they do not change over time as more credentials are revoked.

```
<abc:RevocationAuthorityParameters Version=”1.0”>
  <abc:ParametersUID>xs:anyURI</abc:ParametersUID>
  <abc:RevocationMechanism>xs:anyURI</abc:RevocationMechanism>
  <abc:RevocationInfoReference ReferenceType=”xs:anyURI”>
    …
</abc:RevocationInfoReference>?
  <abc:NonRevocationEvidenceReference ReferenceType=”xs:anyURI”>
    …
</abc:NonRevocationEvidenceReference>?
  <abc:NonRevocationEvidenceUpdateReference ReferenceType=”xs:anyURI”>
    …
</abc:NonRevocationEvidenceUpdateReference>?
<abc:CryptoParams>…</CryptoParams>?
```

```
</abc:RevocationAuthorityParameters>
```

`/abc:RevocationAuthorityParameters`

This element contains the public parameters of the Revocation Authority

`/abc:RevocationAuthorityParameters/@Version`

This attribute indicates the version of this specification. The value MUST be "1.0".

`/abc:RevocationAuthorityParameters/abc:ParametersUID`

This element contains a unique identifier for these Revocation Authority parameters.

`/abc:RevocationAuthorityParameters/RevocationMechanism`

This attribute indicates the mechanism or algorithm used to revoke credentials.

`/abc:RevocationAuthorityParameters/abc:RevocationInfoReference`

This optional element contains a reference to the endpoint where the most current public revocation information corresponding to these parameters can be obtained.

`/abc:RevocationAuthorityParameters/abc:NonRevocationEvidenceReference`

This optional element contains a reference to the endpoint with the information about how to obtain the (possibly private) user-specific non-revocation evidence object.

`/abc:RevocationAuthorityParameters/abc:NonRevocationEvidenceUpdateReference`

This optional element contains a reference to the endpoint the most current information for updating the non-revocation evidence can be obtained.

`/abc:RevocationAuthorityParameters/abc:RevocationInfoReference/@ReferenceType`

This attribute indicates the type of reference to the revocation information endpoint.

`/abc:RevocationAuthorityParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters that are needed to verify the Revocation Information. The content of this element is defined in an external profile based on the value of the `abc:RevocationMechanism` element.

4.3.2 Revocation Information

A Revocation Authority regularly publishes the most recent revocation information, allowing Users to prove and Verifiers to ensure that the credentials used to generate a presentation token have not been revoked. Contrary to the Revocation Authority parameters, the revocation information changes over time, e.g., at regular time intervals, or whenever a new credential is revoked.

The Revocation Authority publishes the revocation information using the artifact described below. How this artifact is protected (authenticated) is application specific; e.g., it could be included in a XML-signed document or provided as part of some metadata retrievable from a trusted source.

```
<abc:RevocationInformation Version="1.0">
  <abc:InformationUID>xs:anyURI</abc:InformationUID>
<abc:RevocationAuthorityParametersUID>
  xs:anyURI
  </abc:RevocationAuthorityParametersUID>
<abc:Created>xs:dateTime</abc:Created>?
```

```
<abc:Expires>xs:dateTime</abc:Expires>?
  <abc:CryptoParams>…</abc:CryptoParams>
</abc:RevocationInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:RevocationInformation`

This element contains the current revocation information, as published by the Revocation Authority. At each update of the revocation information, a new `abc:RevocationInformation` element is generated.

`/abc:RevocationInformation/@Version`

This attribute indicates the version of this specification. The value MUST be "1.0".

`/abc:RevocationInformation/abc:InformationUID`

This element contains the unique identifier of the revocation information. This identifier is different for each version of the revocation information, i.e., a new URI is used at every update.

`/abc:RevocationInformation/abc:RevocationAuthorityUID`

This element contains the identifier of the parameters of the revocation authority that published the revocation information.

`/abc:RevocationInformation/abc:Created`

This optional element contains the date and time when the revocation information was updated or first published.

`/abc:RevocationInformation/abc:Expires`

This optional element contains the date and time until when the revocation information is valid.

`/abc:IssuerParameters/abc:CryptoParams`

This element describes the set of public cryptographic parameters needed to verify whether a credentials is still valid. (The content of this element is defined in an external profile based on the value of the `@RevocationMechanism` attribute specified in the referenced `abc:Revocation AuthorityParameters` element)

4.3.3 Non-Revocation Evidence

The exact details of how and when the non-revocation evidence is created and updated vary greatly among the different revocation mechanisms. We therefore simply define an artifact that acts as a wrapper for a message in a (possibly multi-legged) evidence creation or update protocol. These messages are sent to and received as a response from the evidence creation and update endpoints specified in the Revocation Authority parameters.

```
<abc:RevocationMessage Context="…">
<abc:RevocationAuthorityParametersUID>
  xs:anyURI
  </abc:RevocationAuthorityParametersUID>
  <abc:CryptoParams>…</abc:CryptoParams>
</abc:RevocationMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:RevocationMessage/@Context`

This attribute contains a unique identifier for this protocol session, so that the different flows in the protocol session can be linked together. The request MUST contain a Context attribute. The revocation authority MUST reject requests with context values already in use.

`/abc:RevocationMessage/abc:RevocationAuthorityParametersUID`

This element contains the identifier of the parameters of the revocation authority that creates the non-revocation evidence information.

`/abc:RevocationMessage/abc:CryptoParams`

This element describes the mechanism-specific (cryptographic) parameters needed to obtain the non-revocation evidence information for building or updating the evidence.

4.4 Presentation

The user agent can create presentation tokens using one or more credentials in its possession. The verifier can optionally insist that all credentials used to generate the token are bound to the same user (i.e., to the same user secret) or device.

In a typical ABC presentation interaction, the user first requests access to a protected resource, upon which the verifier sends a presentation policy that describes which credentials the user should present to obtain access. The user agent then checks whether it has the necessary credentials to satisfy the Verifier's presentation policy, and if so, generates a presentation token containing the appropriate cryptographic evidence.

Upon receiving the presentation token, the verifier checks that the cryptographic evidence is valid for the presented credentials and checks that the token satisfies the presentation policy. If both tests succeed, it grants access to the resource.

4.4.1 Presentation Policy

The verifier's policy describes the class of presentation tokens that it will accept. It is expressed by means of a `abc:PresentationPolicyAlternatives` element, with the following schema:

```
<abc:PresentationPolicyAlternatives Version=”1.0”>
<abc:PresentationPolicy EnforceSameUserBinding=”xs:boolean”
 EnforceSameDeviceBinding=”xs:boolean" PolicyUID=”xs:anyURI”?>
    <abc:Message>
      <abc:Nonce>…</abc:Nonce>?
      <abc:ApplicationData>…</abc:ApplicationData>?
    </abc:Message>?
    <abc:Pseudonym Exclusive=”xs:boolean”? Scope=”xs:string”
     Established=”xs:boolean”?>*
    <abc:Credentials>
      <abc:Credential Alias=”xs:anyURI”>
        <abc:CredentialSpecAlternatives>
          <abc:CredentialSpecUID>…</abc:CredentialSpecUID>+
        </abc:CredentialSpecAlternatives>
        <abc:IssuerAlternatives>
```

```
          <abc:IssuerParametersUID
           RevocationInformationUID=”xs:anyURI”?>
            …
          </abc:IssuerParametersUID>+
        </abc:IssuerAlternatives>
        <abc:Attributes>
          <abc:Attribute AttributeType=”xs:anyURI”
           DataHandlingPolicy=”xs:anyURI”?>
          ( <abc:InspectorAlternatives>
              <abc:InspectorPublicKeyUID>…</abc:InspectorPublicKeyUID>+
            </abc:InspectorAlternatives>
            <abc:InspectionGrounds>…</abc:InspectionGrounds>
          )?
            <abc:RevocationParametersUID>…
            </abc:RevocationParametersUID>
            <abc:AcceptableAttributeValues>
              <abc:AttributeValue>…<abc:AttributeValue>+
            <abc:AcceptableAttributeValues>?
          </abc:Attribute>+
        </abc:Attributes>?
      </abc:Credential>+
    </abc:Credentials>?
    <abc:AttributePredicates>
      <abc:AttributePredicate Function=”xs:anyURI”>
      ( <abc:Attribute CredentialAlias=”xs:anyURI”
         AttributeType=”xs:anyURI” DataHandlingPolicy=”xs:anyURI”?/>
        |
        <abc:ConstantValue>…</abc:ConstantValue>
      )+
      </abc:AttributePredicate>+
    </abc:AttributePredicates>?
  </abc:PresentationPolicy>+
</abc:PresentationPolicyAlternatives>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:PresentationPolicyAlternatives`

This element contains a presentation policy, which may contain multiple policy alternatives as child elements. The presented token must satisfy at least one of the specified policies.

`/abc:PresentationPolicyAlternatives/@Version`

This attribute indicates the token version number, it MUST be “1.0”.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy`

This element contains one policy alternative.

`…/abc:PresentationPolicy/@EnforceSameUserBinding`

If the policy requires multiple credentials with user binding to be presented, this attribute being set to true indicates that these credentials must be underlain by the *same* user secret. Insisting credentials to be bound to the same user secret limits users from sharing credentials. Moreover, if the policy requires one or more pseudonyms to be included in the token, this attribute being set to true also insists that all pseudonyms be underlain by the same user secret as the user-bound credentials as well. See Sections 2.3-2.4 for more information on user binding and pseudonyms.

If the policy involves a mixture of credentials, some of which support user binding and some of which don't, then this attribute being set to true indicates that those credentials that support user binding have to be underlain by the same user secret.

`…/abc:PresentationPolicy/@EnforceSameDeviceBinding`

If the policy requires multiple credentials with device binding to be presented, this attribute being set to true indicates that these credentials must be bound to the same device. See Section 2.5 for more information on device binding.

If the policy involves a mixture of credentials, some of which support device binding and some of which don't, then this attribute being set to true indicates that those credentials that support device binding have to be bound to the same device.

`…/abc:PresentationPolicy/@PolicyUID`

This attribute assigns a unique identifier to this presentation policy that can be referenced from presentation tokens that satisfy the policy.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Message`

This optional element specifies a message to be authenticated (signed) by the private key of each credential in the token.

`…/abc:PresentationPolicy/abc:Message/abc:Nonce`

This optional element contains a random nonce.

`…/abc:PresentationPolicy/abc:Message/abc:ApplicationData`

This optional element can contain any application-specific data. The contained data MAY be human readable, depending on the application, and displayed to the user.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Pseudonym`

When present, this optional element indicates that a pseudonym must be presented with the presentation token. If this policy does not involve any credentials to be presented, then a verifiable pseudonym must be presented. Otherwise, a certified pseudonym associated to the presented credentials must be presented. See Section 2.4 for more information on pseudonyms.

`…/abc:PresentationPolicy/abc:Pseudonym/@Scope`

This attribute indicates a string to which the pseudonym is associated. The user agent is assumed to maintain state information to keep track of which pseudonym it previously used for which scope. There can be multiple verifiable or certified pseudonyms associated to the same scope string, but a scope-exclusive pseudonym is guaranteed to be unique with respect to the scope string and the user secret. In the former case, the scope string is merely a hint to the user agent which of its stored pseudonyms can be reused in the presentation token, or to which scope string it should associate a newly created pseudonym. In the latter case, the scope string uniquely determines the pseudonym that needs to be used. The scope string MAY encode an identifier of the verifier and/or of the requested resource. See Section 2.4 for more information on the use of pseudonyms.

`…/abc:PresentationPolicy/abc:Pseudonym/@Exclusive`

When present and set to true, this attribute indicates that a scope-exclusive pseudonym is to be presented with the token. The value of the `@Scope` attribute determines the scope with respect to which the pseudonym must be generated. See Section 2.4 for more information on scope-exclusive pseudonyms.

`…/abc:PresentationPolicy/abc:Pseudonym/@Established`

When set to true, this attribute indicates that the pseudonym to be presented by the User must re-authenticate under a pseudonym that was previously established with the Verifier. When set to false or when not present, this attribute indicates that the User may establish a new pseudonym in the presentation token.

`/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Credentials`

This optional element specifies the set of credentials to be used in the generation of the presentation token. Omitting this element may be useful, for example, when the user can obtain access by merely presenting an existing verifiable pseudonym.

`…/abc:PresentationPolicy/abc:Credentials/abc:Credential`

This element specifies a single credential that has to be used in the generation of the token.

`…/abc:PresentationPolicy/abc:Credentials/abc:Credential/@Alias`

This optional attribute creates an alias for this credential to refer to attributes from this credential in attribute predicates. See the `…/abc:PresentationPolicy/abc:AttributePredicates` element.

`…/abc:Credentials/abc:Credential/abc:CredentialSpecAlternatives`

This element contains a list of credential specifications. The issued credential used to instantiate this credential in the presentation token must adhere to one of the listed credential specifications.

`…/abc:Credential/abc:CredentialSpecAlternatives/abc:CredentialSpecUID`

This element contains one credential specification identifier that can be used to instantiate this credential in the presentation token.

`…/abc:Credentials/abc:Credential/abc:IssuerAlternatives`

This element contains a list of identifiers for issuer parameters UID. The issued credential used to instantiate this credential in the presentation token must be issued under one of the listed issuer parameters.

`…/abc:Credential/abc:IssuerAlternatives/abc:IssuerParametersUID`

This element contains one issuer parameters identifier that is accepted for this credential in the presentation token.

This specification defines two dedicated values for the issuer parameters:

- The value `http://abc4trust.eu/wp2/issuerparameters/unsigned` indicates that the attribute values in this credential are self-claimed, without any form of authentication by either an external issuer or the user herself.
- The value `http://abc4trust.eu/wp2/issuerparameters/pseudonymously-self-signed` indicates that the attribute values in this credential are self-claimed and signed under the pseudonym of the user provided in the same presentation token. This value can only occur when the presentation policy contains a `/abc:PresentationPolicyAlternatives/abc:PresentationPolicy/abc:Pseudonym` element.

`…/abc:IssuerAlternatives/abc:IssuerParametersUID/@RevocationInformationUID`

If the issuer parameters referred to in this element specify an Issuer-driven Revocation Authority, i.e., if the referred `abc:IssuerParameters` element contains an `abc:RevocationParametersUID` child element, then this optional XML attribute can indicate for which version of the revocation information the presented token must be valid. By specifying the current revocation information identifier in the presentation policy, the User does not have to get in touch with the Revocation Authority to check whether her non-revocation evidence information is still up to date, thereby avoiding a possible source of linkability.

`…/abc:PresentationPolicy/abc:Credentials/abc:Credential/abc:Attributes`

This element lists the attributes from this credential that have to be revealed in the presentation token, either to the verifier itself, or to an external inspector.

`…/abc:Credentials/abc:Credential/abc:Attributes/abc:Attribute`

This element specifies one attribute of this credential that has to be revealed in the presentation token, either to the verifier itself, or to an external inspector.

Even though there are no syntactical restrictions imposing this, presentation policies SHOULD NOT request to reveal the value of the revocation handle (with attribute type `http://abc4trust.eu/wp2/abcschemav1.0/revocationhandle`), as doing so enables Verifiers to link presentations tokens generated with the same credential. If necessary, inspection can be used to only reveal the value of the revocation handle under specific circumstances.

`…/abc:Credentials/abc:Credential/abc:Attributes/abc:Attribute/@AttributeTyp e`

This attribute specifies the type of the credential attribute of which the value must be revealed in the presentation token. If multiple credential specifications are allowed for this credential (i.e., if multiple `abc:CredentialSpecUID` elements are listed in the `abc:CredentialSpecAlternatives` child element of the ancestor `abc:Credential` element), then the specified attribute type MUST occur in all listed credential specifications.

For each credential and each attribute type, there MUST be at most one `abc:Attribute` element without `abc:InspectorAlternatives` child element. Likewise, for each credential and each attribute type, there MUST be at most one `abc:Attribute` element with the same `abc:InspectionGrounds` child element.

`…/abc:Credential/abc:Attributes/abc:Attribute/@DataHandlingPolicy`

This XML attribute can be used to refer to an external data handling policy describing how the Verifier will treat the revealed attribute value once it is received. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:InspectorAlternatives`

This optional element lists a number of inspector public key identifiers. When present, this element indicates that the value of this attribute does not have to be revealed to the verifier, but must be encrypted under one of the listed inspector public keys. See Section 2.6 for more details on revealing attributes to an inspector.

`…/abc:Attribute/abc:InspectorAlternatives/abc:InspectorPublicKeyUID`

This element contains one identifier of an inspector public key under which the attribute value can be encrypted.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:InspectionGrounds`

This optional element contains a string describing the valid grounds or circumstances under which the inspector can be asked to decrypt the attribute value or circumstances. This element must be present whenever a sibling `abc:InspectorAlternatives` element is present. See Section 2.6 for more details on revealing attributes to an inspector.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:RevocationParametersUID`

This optional element contains the UID of a revocation authority parameters. The User needs to provide a proof that the attribute value is not revoked according to any of the specified set of parameters.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:AcceptableAttributeValues`

This optional element lists a number of attribute values for a revealed attribute. When present, the attribute value revealed in the presentation token must be equal to one of the listed values.

`…/abc:Attribute/abc:AcceptableAttributeValues/abc:AttributeValue`

This element specifies one permitted value for the revealed attribute.

`…/abc:PresentationPolicy/abc:AttributePredicates`

This optional element specifies a list of predicates that have to hold over the attributes of the credentials listed in the sibling `abc:Credentials` element. To satisfy the policy, the presentation token must for each of the listed predicates either prove (in a data-minimizing way) that the credential attributes satisfy the specified predicate, or must reveal the value of the involved attribute(s) so that the verifier can check whether the predicate is satisfied.

`…/abc:PresentationPolicy/abc:AttributePredicates/abc:AttributePredicate`

This element specifies a predicate that must hold over the attribute values. The child elements are the ordered list of arguments of the predicate.

`…/abc:AttributePredicates/abc:AttributePredicate/@Function`

This attribute specifies the boolean function for this predicate. See Section 4.5.3 for a list of supported functions and their implications on the list of arguments in the child elements.

`…/abc:AttributePredicate/abc:Attribute`

This element specifies a reference to a credential attribute that is to be used as an argument of the predicate.

`…/abc:AttributePredicate/abc:Attribute/@CredentialAlias`

This attribute specifies the alias of the credential from which the attribute must be used. The specified alias MUST also occur as an Alias attribute in an `abc:Credential` element within the ancestor `abc:PresentationPolicy` element.

`…/abc:AttributePredicate/abc:Attribute/@AttributeType`

This attribute refers to the attribute within the credential that is to be used as an argument in the predicate.

`…/abc:AttributePredicate/abc:Attribute/@DataHandlingPolicy`

This XML attribute can be used to refer to an external data handling policy describing how the Verifier will treat the information that the attribute value satisfies the specified predicate. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`…/abc:AttributePredicate/abc:ConstantValue`

This element contains a constant value that is to be used as an argument in the predicate. The data type of the argument depends on the function of the predicate. We refer to Section 4.5.3 for a list of supported functions and the data types of their arguments.

4.4.2 Presentation Token

The presentation of one or multiple credentials results in a presentation token that is sent to the verifier. The syntax for the element is:

```
<abc:PresentationToken Version=”1.0”>
 <abc:PresentationTokenDescription
  EnforceSameUserBinding=”xs:boolean”
  EnforceSameDeviceBinding=”xs:boolean” PolicyUID=”xs:anyURI”
  TokenUID=”xs:anyURI”?>
  <abc:Message>
    <abc:Nonce>…</abc:Nonce>?
    <abc:ApplicationData>…</abc:ApplicationData>?
  </abc:Message>?
  <abc:Pseudonym Scope=”xs:string”? Exclusive=”xs:boolean”?>
      <abc:PseudonymValue>…</abc:PseudonymValue>
  </abc:Pseudonym>*
  <abc:Credentials>
    <abc:Credential Alias=”xs:anyURI”>
      <abc:CredentialSpecUID>…</abc:CredentialSpecUID>
      <abc:IssuerParametersUID>…</abc:IssuerParametersUID>
      <abc:RevocationInformationUID>
        …
      </abc:RevocationInformationUID>?
      <abc:Attributes>
        <abc:Attribute AttributeType=”xs:anyURI”
         DataHandlingPolicy=”xs:anyURI”?>
        ( <abc:InspectorPublicKeyUID>…</abc:InspectorPublicKeyUID>
          <abc:InspectionGrounds>…</abc:InspectionGrounds>
        )?
          <abc:RevocationInformationUID>
            …
          </abc:RevocationInformationUID>*
          <abc:AttributeValue>…</abc:AttributeValue>
        </abc:Attribute>+
      </abc:Attributes>?
      <abc:CryptoEvidence>...</abc:CryptoEvidence>
    </abc:Credential>+
  </abc:Credentials>?
  <abc:AttributePredicates>
    <abc:AttributePredicate Function=”xs:anyURI”>
    ( <abc:Attribute CredentialAlias=”xs:anyURI”
       AttributeType=”xs:anyURI”
       DataHandlingPolicy=”xs:anyURI”?/>
      |
      <abc:ConstantValue>…</abc:ConstantValue>
    )+
    </abc:AttributePredicate>+
  </abc:AttributePredicates>?
  </abc:PresentationTokenDescription>
  <abc:CryptoEvidence>…</abc:CryptoEvidence>
</abc:PresentationToken>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:PresentationToken`

This element contains a presentation token.

`/abc:PresentationToken/@Version`

This attribute indicates the token version number, it MUST be “1.0”.

`/abc:PresentationTokenDescription`

This element contains a technology-agnostic description of the revealed   information.

`/abc:PresentationToken/PresentationTokenDescription/@EnforceSameUserBinding`

If the presentation token involves multiple credentials with user binding, this attribute being set to true indicates that these credentials are underlain by the *same* user secret. Moreover, if the token includes a pseudonym, then this attribute being set to true also insists that the pseudonym be underlain by the same user secret. See Sections 2.3-2.4 for more information on user binding and pseudonyms.

`…/abc:PresentationPolicy/@PolicyUID`

This attribute refers to the UID of the presentation policy that this token satisfies.

`…/abc:PresentationPolicy/@TokenUID`

This optional attribute assigns a unique identifier to this presentation token.

`…/abc:PresentationTokenDescription/abc:Message`

This optional element specifies a message that is authenticated (signed) by the private key of each credential in the token.

`…/abc:PresentationTokenDescription/abc:Message/abc:Nonce`

This optional element contains a random nonce that is to be signed by a presentation token satisfying this policy. The nonce is generated by the Issuer and prevents replay attacks.

`…/abc:PresentationTokenDescription/abc:Message/abc:ApplicationData`

This optional element can contains data of type string.

`…/abc:PresentationTokenDescription/abc:Pseudonym`

When present, this element indicates that a pseudonym is presented with the presentation token. If this policy does not involve any credentials, then this is a verifiable pseudonym, otherwise it is a certified pseudonym associated to the presented credentials. See Section 2.4 for more information on pseudonyms.

`…/abc:PresentationTokenDescription/abc:Pseudonym/@Scope`

This optional attribute indicates that the presented pseudonym is for a specific scope (e.g., a resource identifier) See Section 2.4 for more information on the use of pseudonyms. The user agent is assumed to maintain state information to keep track of which pseudonym it previously used for which scope.

`…/abc:PresentationTokenDescription/abc:Pseudonym/@Exclusive`

When present, this attribute indicates that a scope-exclusive pseudonym is presented with the token. The value of the `@Scope` attribute determines the scope with respect to which the pseudonym was generated. See Section 2.4 for more information on scope-exclusive pseudonyms.

`…/abc:PresentationTokenDescription/abc:Pseudonym/abc:PseudonymValue`

This element contains the value of the pseudonym encoded as content of type `xs:base64Binary`.

`…/abc:PresentationTokenDescription/abc:Credentials`

This optional element specifies credentials that are presented in this token. If the token contains no `abc:Credentials` element but does contain an `abc:Pseudonym`, then this presentation token merely proves knowledge of the user secret underlying the pseudonym.

`…/abc:PresentationTokenDescription/abc:Credentials/abc:Credential`

This element specifies a single credential that is presented in this token.

`…/abc:PresentationTokenDescription/abc:Credentials/abc:Credential/@Alias`

This optional attribute defines an alias for this credential to refer to attributes from this credential in attribute predicates. See the `/abc:PresentationToken/abc:AttributePredicates` element.

`…/abc:Credentials/abc:Credential/abc:CredentialSpecUID`

This element contains the credential specification identifier of the presented credential.

`/abc:Credentials/abc:Credential/abc:IssuerParametersUID`

This element contains the issuer public key identifier of the presented credential.

`…/abc:Credentials/abc:Credential/abc:RevocationInformationUID`

This optional element contains an identifier of the revocation information with respect to which the presented credential is proved to be non-revoked. The revocation information referenced here corresponds to the issuer-driven revocation parameters referenced from the issuer parameters; see the `/abc:PresentationToken/abc:Credentials/abc:Credential/abc:Attributes/abc:Attribute/abc:RevocationInformationUID` element for verifier-driven revocation.

When verifying the token, the verifier has to independently obtain the current revocation information using the mechanism specified by the revocation authority parameters referenced in the `IssuerParameters`. It is up to the verifier to check that the revocation information UID referenced in this element is indeed the most recent one.

`…/abc:Credentials/abc:Credential/abc:Attributes`

This element lists the attributes from this credential that are revealed by this presentation token, either in the clear to the verifier itself, or encrypted to an external inspector.

`…/abc:Credentials/abc:Credential/abc:Attributes/abc:Attribute`

This element specifies one attribute of this credential that is revealed in the presentation token.

`…/abc:Credential/abc:Attributes/abc:Attribute/@AttributeType`

This attribute specifies the type of the credential attribute of which the value is revealed.

There MUST be at most one `abc:Attribute` element without `abc:InspectorPublicKeyUID` child element per credential and per attribute type. Also, for `abc:Attribute` elements with an `abc:InspectorPublicKeyUID` child element, there MUST be at most one `abc:Attribute` element per credential and per attribute type with the same `abc:InspectionGrounds` child element.

`…/abc:Credential/abc:Attributes/abc:Attribute/@DataHandlingPolicy`

This optional XML attribute can be used to refer to an external data handling policy that the Verifier has to adhere to concerning the revealed attribute value. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:InspectorPublicKeyUID`

This optional element contains the identifier of the inspector public key under which the attribute value is encrypted.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:InspectionGrounds`

This optional element contains a string describing the valid grounds or circumstances under which the inspector can be asked to decrypt the attribute value or circumstances. This element must be present whenever a sibling `abc:InspectorPublicKeyUID` element is present. See Section 2.6 for more details on revealing attributes to an inspector.

`…/abc:Credential/abc:Attributes/abc:Attribute/abc:RevocationInformationUID`

This optional element contains an identifier of revocation information with respect to which the presented credential is proved to be non-revoked. The revocation information referenced here corresponds to the verifier-driven revocation parameters mentioned in the verifier's presentation policy; see the `/abc:PresentationToken/abc:Credentials/abc:Credential/abc:RevocationInformationUID` element for issuer-driven revocation. When verifying the token, the verifier has to independently obtain the current revocation information using the mechanism specified by the revocation authority parameters referenced in the presentation policy. It is up to the verifier to check that the revocation information UID referenced in this element is indeed the most recent one.

If the verifier's presentation policy contained multiple (references to) revocation parameters for this credential attribute, then the token must contain a `abc:RevocationInformationUID` element for each of the parameters referenced in the presentation token.

`…/abc:Attributes/abc:Attribute/abc:AttributeValue`

This element specifies the value of the revealed attribute. When encrypted to an inspector, this element MAY contain data of type `xs:base64Binary` representing the ciphertext for the encrypted attribute. However, there is no guarantee that this data by itself is decryptable by the inspector. When requesting decryption of an attribute, the complete presentation token must always be sent to the inspector.

`…/abc:Credentials/abc:Credential/abc:CryptoEvidence`

This element contains cryptographic evidence for this particular credential. There is no guarantee that this evidence is verifiable by itself, only the presentation token as a whole can be verified.

`…/abc:PresentationTokenDescription/abc:AttributePredicates`

This optional element specifies a list of predicates over the attributes of the listed credentials that are all guaranteed to be true by this token.

`…/abc:AttributePredicates/abc:AttributePredicate`

This element specifies a single predicate that is guaranteed to hold by this token. The child elements are the ordered list of arguments of the predicate.

`…/abc:AttributePredicates/abc:AttributePredicate/@Function`

This attribute specifies the boolean function for this predicate. See Section 4.5.3 for a list of supported functions and their implications on the list of arguments in the child elements.

`…/abc:AttributePredicate/abc:Attribute`

This element specifies a reference to a credential attribute that is used as an argument of the predicate.

`…/abc:AttributePredicate/abc:Attribute/@CredentialAlias`

This attribute specifies the alias of the credential from which the attribute is used. The specified value MUST also occur as an Alias attribute in an `abc:Credential` element within this `abc:PresentationToken`.

`…/abc:AttributePredicate/abc:Attribute/@AttributeType`

This attribute refers to the exact attribute within the credential that is used as an argument in the predicate.

`…/abc:AttributePredicate/abc:Attribute/@DataHandlingPolicy`

This optional XML attribute can be used to refer to an external data handling policy that the Verifier has to adhere to with respect to the information that the attribute value satisfies the specified predicate. The data handling policy may be human-readable and/or machine-readable. The specification of a data handling policy schema is outside of the scope of this document.

`…/abc:AttributePredicate/abc:ConstantValue`

This element contains a constant value that is used as an argument in the predicate. The data type of the argument depends on the function of the predicate. We refer to Section 4.5.3 for a list of supported functions and the data types of their arguments.

`/abc:PresentationToken/abc:CryptoEvidence`

This element contains the cryptographic evidence for the presentation token.

4.4.3 Functions for Use in Predicates

When evaluating predicates over attributes in presentation policies and presentation tokens, the following list of function URIs from [XACML20] for (in)equality testing of different data types MUST be supported. We refer to [XACML20, Appendix A] for the semantics of these functions and the data types of their arguments. In order to prove predicates over credential attributes, the involved attributes MUST use the same encoding (see Section 4.2.1).

```
urn:oasis:names:tc:xacml:1.0:function:string-equal
urn:oasis:names:tc:xacml:1.0:function:boolean-equal
urn:oasis:names:tc:xacml:1.0:function:integer-equal
urn:oasis:names:tc:xacml:1.0:function:date-equal
urn:oasis:names:tc:xacml:1.0:function:time-equal
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:integer-less-than
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:date-greater-than
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal
urn:oasis:names:tc:xacml:1.0:function:date-less-than
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal
```

Moreover, this specification defines the following list of new functions for inequality testing.

```
urn:abc4trust:1.0:function:string-not-equal
urn:abc4trust:1.0:function:boolean-not-equal
urn:abc4trust:1.0:function:integer-not-equal
urn:abc4trust:1.0:function:date-not-equal
urn:abc4trust:1.0:function:time-not-equal
urn:abc4trust:1.0:function:dateTime-not-equal
urn:abc4trust:1.0:function:anyURI-not-equal
```

For *type* being one of *string, boolean, integer, date, time, dateTime,* or *anyURI*, the semantics of function `urn:abc4trust:1.0:function:type-not-equal` is defined as follows. The function SHALL take two arguments of data-type *http://www.w3.org/2001/XMLSchema#type* and SHALL return an *http://www.w3.org/2001/XMLSchema#boolean*. The function SHALL return *true* if and only if the application of the corresponding function `urn:oasis:names:tc:xacml:1.0:function:type-equal` evaluated on the same arguments returns *false*. Otherwise, it SHALL return *false*.

Finally, this specification defines the following list of functions for testing equality against a list of candidate values.

```
urn:abc4trust:1.0:function:string-equal-oneof
urn:abc4trust:1.0:function:boolean-equal-oneof
urn:abc4trust:1.0:function:integer-equal-oneof
urn:abc4trust:1.0:function:date-equal-oneof
urn:abc4trust:1.0:function:time-equal-oneof
urn:abc4trust:1.0:function:dateTime-equal-oneof
urn:abc4trust:1.0:function:anyURI-equal-oneof
```

For *type* being one of *string, boolean, integer, date, time, dateTime,* or *anyURI*, the semantics of function `urn:abc4trust:1.0:function:type-equal-oneof` is defined as follows. The function SHALL take two or more arguments of data-`type http://www.w3.org/2001/XMLSchema#type` and SHALL return an `http://www.w3.org/2001/XMLSchema#boolean`. The function SHALL return *true* if and only if the application of the corresponding function `urn:oasis:names:tc:xacml:1.0:function:type-equal` evaluated on the first argument and one of the arguments other than the first returns *true*. Otherwise, it SHALL return *false*.

4.5 Issuance

Issuance of Privacy-ABCs is an interactive process between the User and the Issuer, possibly involving multiple exchanges of messages. This document specifies the contents, encoding, and processing of the messages; an application needs to define how to exchange them, e.g., by embedding them in existing messaging protocols.[3]

[3] For example, WS-Trust [WS-Trust14] specifies an issuance challenge-response pattern that can be used to carry the ABC issuance messages, embedding them in `RequestSecurityToken` and `RequestSecurityTokenResponse` messages.

An overview of a typical issuance interaction is given in Figure 4.1. The User initiates the interaction by sending an issuance request to the Issuer, optionally specifying the requested credential specification UID.

In the simplest case, the credential is issued “from scratch”, i.e., without relation to any existing credentials. Even in this case, the issuance protocol may consist of multiple exchanges of issuance messages.

In a more advanced setting, the new credential that is being issued may carry over attribute values, the user secret or the device secret from credentials that the User already owns, or may require attributes values to be generated jointly at random. We refer to Section 2.7 for more details on the possibilities of advanced issuance protocols.

In the advanced setting, the issuer responds to the initial request with its issuance policy, which specifies which issuance token the user must present in order to obtain the requested token, which features of existing credentials will be carried over to the new credential, and which attributes will be generated jointly at random. The user responds with an issuance token. Then, a number of interaction rounds may take place to perform the cryptographic issuance protocol. At the end of these rounds, the Issuer sends the final message allowing the User to construct the issued credential.

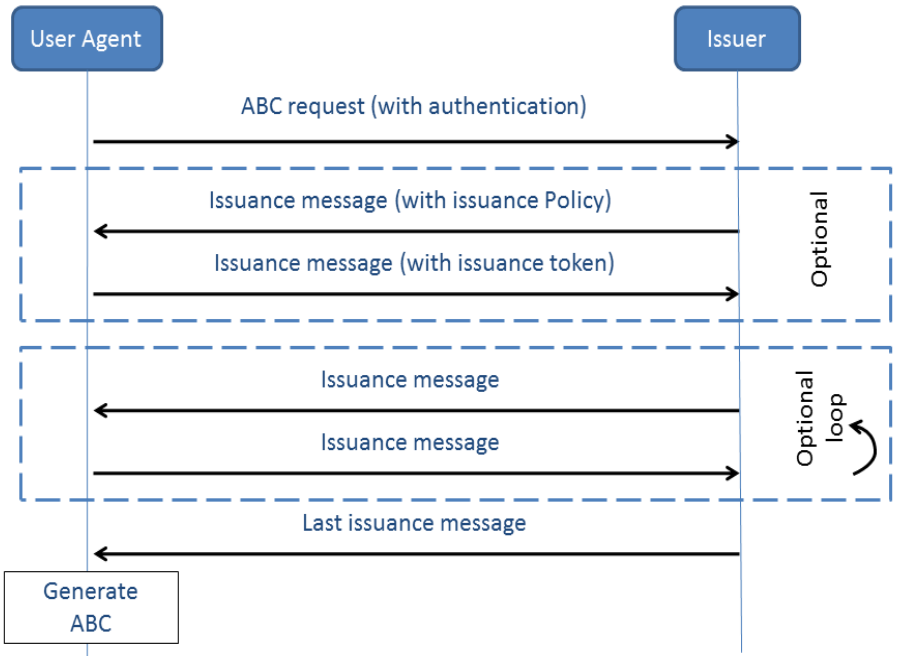
User Agent
Issuer
ABC request (with authentication)
Issuance message (with issuance Policy)
Issuance message (with issuance token)
Optional
Issuance message
Issuance message
Optional loop
Last issuance message
Generate ABC

Figure 4.1 - Issuance of Privacy-ABCs

Some notes:

- The endpoint to contact, and its authentication requirements, are application specific. The issuance protocol SHOULD be done over a secure channel to protect the confidentiality of the attribute values.

- Since the exchange is multi-legged, the parties must keep the cryptographic state of each issuance instance between the message exchanges.

User authentication is out of scope of this document. Authentication information MAY be provided along the issuance messages.

4.5.1 Issuance Policy

Optionally, the Issuer may respond to the User's initial request by sending the issuance policy. In an issuance policy, the Issuer describes which credentials he will issue based on which issuance token presented by the User. The newly issued credential can "carry over" certain features from the existing credentials used in generating the issuance token, without revealing these features to the Issuer. Namely, the newly issued credential can be bound to the same User, to the same device, or to the same revocation handle as one of the existing credentials. Also, attribute values in the new credential can be carried over from attributes in the existing credentials, without the Issuer being able to see these attribute values.

Any message that will be exchanged in the course of an issuance protocol is wrapped in an `IssuanceMessage`. That includes the issuance policy and issuance token (if requested by the issuer), as well as the subsequent interactions between the User and Issuer to execute the cryptographic protocol. To allow the linkage of the different legs of a protocol, each message includes a Context attribute, which must have the same value on all legs (including the possible preceding issuance policy/token exchange).

```
<abc:IssuanceMessage Context="…">
  <abc:IssuancePolicy Version="1.0">
    <abc:PresentationPolicy … >  …  </abc:PresentationPolicy>
    <abc:CredentialTemplate>
      <abc:CredentialSpecUID>…</abc:CredentialSpecUID>
      <abc:IssuerParametersUID>…</abc:IssuerParametersUID>
      <abc:UnknownAttributes>
        <abc:UserBinding>
          <SourceCredentialInfo Alias="xs:anyURI">
        </abc:UserBinding>?
        <abc:DeviceBinding >
          <SourceCredentialInfo Alias="xs:anyURI">
        </abc:DeviceBinding>?
        <abc:Attributes>
          <abc:Attribute                    TargetAttributeType="xs:anyURI"
                    JointlyRandom="xs:boolean">
            <SourceCredentialInfo                         Alias="xs:anyURI"
                             AttributeType="xs:anyURI">
          </abc:Attribute>+
        </abc:Attributes>?
      </abc:UnknownAttributes>
    </abc:CredentialTemplate>
  </abc:IssuancePolicy>
</abc:IssuanceMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceMessage`

This element contains a message in a (possibly multi-legged) issuance protocol. In this particular flow, it is the first message in the protocol containing an issuance policy.

`/abc:IssuanceMessage/@Context`

The message MUST contain a unique context attribute chosen by the Issuer to link together the different `IssuanceMessage`s of this instance of the issuance protocol.

`/abc:IssuancePolicy`

This element describes an issuance policy.

`/abc:IssuancePolicy/abc:PresentationPolicy`

This element specifies which token has to be presented by the user in order to be issued a credential. See the `/abc:PresentationPolicyAlternatives/abc:PresentationPolicy` element in Section 4.5.1 for a description of the schema. The main goal of this policy and the issuance token returned in response of it is to carry over features from the existing credentials used to generate the presentation token into the newly issued credential.

Note that the presentation policy can also request for a self-signed of self-stated credential; see the `IssuerParametersUID` element in the `PresentationPolicy` for details. Using this feature, the Issuer can have self-signed and self-claimed attributes to be carried over into the newly issued credential. These attribute values will be visible to the Issuer if the issuance policy explicitly specifies that they must be revealed, or will be invisible to the Issuer otherwise.

`/abc:IssuancePolicy/abc:CredentialTemplate/`

This element provides a template for the to-be-issued credential.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:CredentialSpecUID`

This element contains the unique identifier of the credential specification of the newly issued credential.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:IssuerParametersUID`

This element contains the unique identifier of the issuer parameters of the newly issued credential.

`/abc:IssuancePolicy/abc:CredentialTemplate/abc:UnknownAttributes`

This element specifies the attributes that are unknown to the Issuer and that will either be carried over from another credential or jointly generated at random.

```
…/abc:CredentialTemplate/abc:UnknownAttributes/(abc:UserBinding |
abc:DeviceBinding)
```

This element describes how the user or device binding is established.

```
…/…/abc:UnknownAttributes/(abc:UserBinding |
abc:DeviceBinding)/@JointlyRandom
```

This attribute indicates if the user or device secret is jointly generated at random.

```
…/…/abc:UnknownAttributes/(abc:UserBinding |
abc:DeviceBinding)/SourceCredentialInfo/@Alias
```

This attribute indicates the alias of the presented credential from which to carry-over the user or device secret.

`…/abc:CredentialTemplate/abc:UnknownAttributes/abc:Attributes`

This element describes how the unknown attributes are established.

`…/abc:CredentialTemplate/abc:UnknownAttributes/abc:Attributes/abc:Attribute`

This element describes how a particular unknown attribute is established.

`…/abc:UnknownAttributes/abc:Attributes/abc:Attribute/@TargetAttributeType`

This attribute indicates to which attribute in the to-be-issued credential this template information applies to.

`…/abc:UnknownAttributes/abc:Attributes/abc:Attribute/@JointlyRandom`

This attribute indicates that the target attribute value is jointly generated at random.

`…/abc:Attributes/abc:Attribute/abc:SourceCredentialInfo`

This element contains information about the source credential to transfer the info from.

`…/abc:Attributes/abc:Attribute/abc:SourceCredentialInfo/@Alias`

This attribute indicates the alias of the presented credential from which to carry-over the attribute value.

`…/abc:Attributes/abc:Attribute/abc:SourceCredentialInfo/@AttributeType`

This attribute indicates the attribute type of the presented credential from which to carry-over the attribute value (which could be different than the target attribute type, e.g., from the `LastName` attribute of the `DriverLicense` credential to the `GivenName` attribute of the `StudentCard` credential).

4.5.2 Issuance Token

The User responds with an issuance token, that contains a presentation token and credential template satisfying the issuance policy of the Issuer. In order to satisfy the policy, the credential template in the issuance token must be the same as in the received issuance policy. See Section 4.4.2 for the schema of the presentation token and Section 4.5.1 for the schema of the credential template.

```
<abc:IssuanceMessage Context="…">
  <abc:IssuanceToken Version="1.0">
    <abc:IssuanceTokenDescription>
      <abc:PresentationTokenDescription>
      …
      </abc:PresentationTokenDescription>
      <abc:CredentialTemplate>
     …
      </abc:CredentialTemplate>
    </abc:IssuanceTokenDescription>
    <abc:CryptoEvidence>
     …
    </abc:CryptoEvidence>
  </abc:IssuanceToken>
</abc:IssuanceMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceMessage`

This element contains a message in a (possibly multi-legged) issuance protocol. In this particular flow, it contains an issuance token fulfilling the Issuer’s issuance policy. The issuance policy may have been transmitted through an `IssuanceMessage` in a previous flow, or the User may have obtained the policy in some other way, in which case this is the first flow of the protocol.

`/abc:IssuanceMessage/@Context`

The message MUST contain a unique context attribute and its value MUST match the one from the initial IssuanceMessage containing the issuance policy (if any).

`/abc:IssuanceToken`

This element describes an issuance token.

`/abc:IssuanceToken/@Version`

This attribute indicates the token version number, it MUST be "1.0".

`/abc:IssuanceToken/abc:IssuanceTokenDescription`

This element contains a technology-agnostic description of the revealed information and the new credential.

`…/abc:IssuanceTokenDescription/abc:PresentationTokenDescription`

This element contains a technology-agnostic description of the revealed information.

`…/abc:IssuanceTokenDescription/abc:CredentialTemplate/`

This element provides a template for the to-be-issued credential.

`/abc:IssuanceToken/abc:CryptoEvidence/`

This element provides the cryptographic evidence for the issuance token.

4.5.3 Issuance Messages

The message contents in the remaining flows of the issuance protocol are mechanism-specific and therefore treated as opaque pieces of information that are exchanged between the Issuer and the User.

```
<abc:IssuanceMessage Context="…">
    …
</abc:IssuanceMessage>
```

The following describes the attributes and elements listed in the schema outlined above:

`/abc:IssuanceMessage`

This element contains mechanism-specific cryptographic issuance data.

`/abc:IssuanceMessage/@Context`

The message MUST contain a context attribute and its value MUST match the one from the initial IssuanceMessage (if any).

5 API for Privacy-ABCs

This chapter describes the application programming interfaces (API) of the ABCE layer. Section 5.1 describes the API that the ABCE layer exposes to the upper layers, in particular, to the application layer. Section 5.2 describes the interfaces of the internal components of the ABCE layer and how they interact with each other. Section 5.1 is mainly intended for developers who want to build applications that make use of ABCE technology. Section 5.2 is mainly intended for developers who want to build their own cryptographic ABCE providers and plug them into the framework, and for the ABC4Trust partners who will collaborate on building the ABCE infrastructure.

The interfaces are described in an object-oriented fashion as a list of methods that take input parameters of certain types and that produce an output of a certain return type. The data types of the input and return types either refer to XML artifacts as defined in Chapter 4, to simple XML Schema datatypes such as boolean or string, or to internal data types that remain to be defined. For the latter category, we will sketch in Section 5.3 what information they contain, but do not fix a representation as it is not clear yet whether these types will be encoded in XML or passed as object structures

5.1 External API

This subsection describes the top-level application programming interfaces (API) that the ABCE layer exposes to the upper layers, in particular, the application layer. The information in this section is mainly intended for application programmers who want to build ABCE-enabled applications on top of the ABC4Trust framework.

For ease of integration with applications built on top of our ABCE layer, the actual implementation will offer the top-level ABCE interfaces described below as web services. The descriptions below must therefore be mapped to descriptions in the Web Services Description Language (WSDL). Doing so is straightforward, so for the sake of readability we stick to an object-oriented notation here.

5.1.1 ABCE methods for Users

`setupUser(keylength:integer, mechanism:anyURI)`

> This method, on input a key length (the number of bits in the key) and the identifier of the cryptographic mechanism to be used, generates a cryptographically strong user secret and stores it in trusted storage.

`canBeSatisfied(p:PresentationPolicyAlternatives) → boolean`

> This method, on input a presentation policy `p`, decides whether the credentials in the User's credential store could be used to produce a valid presentation token satisfying the policy `p`. If so, this method returns true, otherwise, it returns false.

`createPresentationToken(p:PresentationPolicyAlternatives) → PresentationToken`

> This method, on input a presentation policy `p`, returns a presentation token that satisfies the policy `p`, or returns an error if no such token could be created. This method will investigate whether the User has the necessary credentials and/or established pseudonyms to create a token that satisfies the policy. If there are multiple ways in which the policy can be satisfied (e.g., by satisfying different alternatives in the policy, or by using different sets of credentials to satisfy one alternative), this method will invoke an identity selection – possibly presented as a user interface (the executable code of which is installed on the User's machine as part of the ABCE framework) – to let the user choose her preferred way of generating the presentation

token. This method will further invoke a user interface to ensure User consent to the information that will be revealed in the created token. If the policy cannot be satisfied (if the `canBeSatisfied` method would have returned false), then the method returns an error.

`createIssuanceToken(m:IssuanceMessage, atts:Attribute[]) → IssuanceMessage`

This method, on input an issuance message `m`, containing an issuance policy `ip`, and a list of attribute type-value pairs `atts`, returns an issuance token that satisfies the issuance policy and that also contains the self-claimed attributes in `atts`, or returns nothing (null) if no such token could be created. The attribute types contained in `atts` must be attribute types that occur in (the credential specification of) the credential to be issued.

This method will investigate whether the User has the necessary credentials and/or established pseudonyms to create an issuance token that satisfies the issuance policy. If there are multiple ways in which the policy can be satisfied (e.g., by using different sets of credentials), this method will invoke an identity selection to choose the preferred way of generating the presentation token. If the policy cannot be satisfied, then the method returns an error.

`issuanceProtocolStep(m:IssuanceMessage) → IssuanceMessage or CredentialDescription`

This method performs one step in an interactive issuance protocol. On input an incoming issuance message `m` obtained from the Issuer, it either returns the outgoing issuance message that is to be sent back to the Issuer, or returns a description of the newly issued credential at successful completion of the protocol. In the former case, the Context attribute of the outgoing message has the same value as that of the incoming message, allowing the Issuer to link the different messages of this issuance protocol.

`updateNonRevocationEvidence()`

This method updates the non-revocation evidence associated to all credentials in the credential store. Calling this method at regular time intervals reduces the likelihood of having to update non-revocation evidence at the time of presentation, thereby not only speeding up the presentation process, but also offering improved privacy as the Revocation Authority is no longer “pinged” at the moment of presentation.

`listCredentials()→ anyURI[]`

This method returns an array of the unique credential identifiers (UIDs) for all credentials available to the Credential Manager.

`getCredentialDescription(creduid:anyURI) → CredentialDescription`

This method returns the description of the credential with the given unique identifier. The unique credential identifier `creduid` is the identifier which was included in the credential description that was returned at successful completion of the issuance protocol.

`deleteCredential(creduid:anyURI) → boolean`

This method deletes the credential with the given identifier from the credential store. If deleting is not possible (e.g. if the refered credential does not exist) the method returns false, and true otherwise.

5.1.2 ABCE methods for Verifiers

`verifyTokenAgainstPolicy(p:PresentationPolicyAlternatives, t:PresentationToken, store:boolean) → PresentationTokenDescription or String[]`

This method, on input a presentation policy `p` and a presentation token t, checks whether the token `t` satisfies the policy `p`, and checks the validity of the cryptographic evidence included in token `t`. If both checks succeed and store is set to true, this method stores the token in a dedicated store and returns a description of the token that includes a unique identifier by means of which the token can later be retrieved from the store. If one of the checks fails, this method returns a list of error messages.

`getToken(tokenuid:anyURI) → PresentationToken`

This method looks up a previously verified presentation token. The unique token identifier tokenuid is the identifier that was included in the token description that was returned by the `PolicyTokenMatcher.verifyToken` method when the token was verified.

`deleteToken(tokenuid:anyURI) → boolean`

This method deletes the previously verified presentation token referenced by the unique identifier tokenuid. It returns true in case of successful deletion, and false otherwise.

5.1.3 ABCE methods for Issuers

`setupSystemParameters(keylength:integer, mechanism:anyURI) → SystemParameters`

This method generates a fresh set of system parameters for the given key length and cryptographic mechanism. Issuers can generate their own system parameters, but can also reuse system parameters generated by a different entity. More typically, a central party (e.g., a standardization body) will generate and publish system parameters for a number of different key lengths that will be used by many Issuers.

`setupIssuerParameters(credspec:CredentialSpecification, syspars:SystemParameters, uid:anyURI, hash:anyURI, revparsuid:anyURI) → IssuerParameters`

This method generates a fresh issuance key and the corresponding Issuer parameters. The issuance key is stored in the Issuer's key store, the Issuer parameters are returned as output of the method. The input to this method specify the credential specification `credspec` of the credentials that will be issued with these parameters, the system parameters `syspars`, the unique identifier `uid` of the generated parameters, the hash algorithm identifier `hash`, and, optionally, the parameters identifier for any Issuer-driven Revocation Authority.

`initIssuanceProtocol(ip:IssuancePolicy, atts:Attribute[]) → (IssuanceMessage, boolean)`

This method is invoked by the Issuer to initiate an issuance protocol based on the given issuance policy `ip` and the list of attribute type-value pairs `atts` to be embedded in the new credential. It returns an `IssuanceMessage` that is to be sent to the User and fed to the `Issuance Manager.issuanceProtocolStep` method on the User's side. The `IssuanceMessage` contains a Context attribute that will be the same for all message exchanges in this issuance protocol, to facilitate linking the different flows of the protocol.

In case of an issuance "from scratch", i.e., for which the User does not have to prove ownership of existing credentials or established pseudonyms, the given issuance policy `ip` merely specifies the credential specification and the issuer parameters for the credential to be issued. In this case, the returned issuance message is the first message in the actual cryptographic issuance protocol.

In case of an "advanced" issuance, i.e., where the User has to prove ownership of existing credentials or pseudonyms to carry over attributes, a user secret, or a device secret, the returned `IssuanceMessage` is simply a wrapper around the issuance policy `ip` with a fresh Context attribute. The returned boolean indicates whether this is the last flow of the issuance protocol. If the `IssuanceMessage` is not the final one, the Issuer will subsequently invoke its `issuanceProtocolStep` method on the next incoming IssuanceMessage from the User.

`issuanceProtocolStep(m:IssuanceMessage) → (IssuanceMessage, boolean)`

This method performs one step in an interactive issuance protocol. On input an incoming issuance message `m` received from the User, it returns the outgoing issuance message that is to be sent back to the User, plus a boolean indicating whether this is the last message in the protocol. The Context attribute of the outgoing message has the same value as that of the incoming message, allowing the Issuer to link the different messages of this issuance protocol.

5.1.4 ABCE methods for Revocation Authorities

```
setupRevocationAuthorityParameters(keylength:integer, mechanism:anyURI,
uid:anyURI, inforef:RevocationInfoReference,
evidenceref:NonRevocationEvidenceReference,
updref:RevocationUpdateReference) → RevocationAuthorityParameters
```

For a given key length and revocation mechanism, this method generates a fresh secret key for the Revocation Authority and corresponding public Revocation Authority parameters, as well as the initial revocation information. The secret key is stored in trusted storage. Also included in the returned Revocation Authority parameters are the given identifier `uid` as well as the endpoints where Users, Verifiers and Issuers can obtain the latest revocation information (`inforef`), initial non-revocation evidence (`evidenceref`), and updates to their non-revocation evidence (`updref`).

`getCurrentRevocationInformation(rparsuid:anyURI) → RevocationInformation`

This method takes as input the unique identifier (UID) of revocation authority parameters `rparsuid` and returns the latest revocation information corresponding to the specified revocation parameters.

`revoke(rparsuid:anyURI, atts:Attribute[]) → RevocationInformation`

This method revokes the attribute values specified by the input parameter `atts` with respect to the revocation parameters specified by their unique identifier `rparsuid`. When `atts` contains multiple attribute type-value pairs, then the combination of these attribute values is revoked, i.e., all credentials that have the combination of attribute values specified in `atts` are revoked. In the special case of Issuer-driven revocation, `atts` contains one attribute value that is the revocation handle, so that only the unique credential with that revocation handle has been revoked.

5.1.5 ABCE methods for Inspectors

```
setupInspectorPublicKey(keylength:integer, mechanism:anyURI, uid:anyURI) →
InspectorPublicKey
```

This method generates a fresh decryption key and corresponding encryption key for the given key length and cryptographic mechanism. It stores the decryption key in the trusted storage and returns the inspector public key with the given identifier `uid`. The identifier associated with the key will be used in presentation/issuance policies as the unique reference to the dedicated Inspector.

`inspect(t:PresentationToken) → Attribute[]`

This method takes as input a presentation token with inspectable attributes and returns the decrypted attribute type-value pairs for which the Inspector has the inspection secret key.

5.2 Internal APIs

This subsection describes the APIs that the different components within the ABCE layer expose to each other, and how they collaborate in a typical flow of invocations. This section is mainly intended for implementers who want to add new cryptographic providers, as well as for the ABC4Trust partners to structure the implementation of the ABC4Trust framework.

The data types of the input and return types either refer to XML artifacts as defined in Chapter 4 to simple XML Schema datatypes such as boolean or string, or to internal data types that remain to be defined. For the latter category, we will sketch in Section 5.3 what information they contain, but do not fix a representation as it is not clear yet whether these types will be encoded in XML or passed as object structures.

We assume that there is a mechanism in place (e.g., through configuration files) to let the different components know how to communicate with each other (e.g., through web services, inter-process communication, or object-oriented method calls), where the credential store can be found, etc.

5.2.1 PolicyCredentialMatcher (User)

`createPresentationToken(p:PresentationPolicyAlternatives) → PresentationToken`

This method returns a presentation token that satisfies the given presentation policy (or more precisely, at least one of them in case p contains several presentation policies) It performs the following steps in doing so.

- Obtain the descriptions of all "relevant" credentials in the credential store. With "relevant", we mean a first selection of credentials based only on the combinations of allowed Issuer parameters and credential specifications. This list is obtained by repetitively calling `CredentialManager.getCredentialDescriptions(issuers, credspecs)` for each credential required by the presentation policy. The parameters *`issuers`* and `credspecs` are the lists of acceptable Issuer parameter UIDs and credential specification UIDs for this credential as specified in the presentation policy. Each returned credential description contains the credential UID, i.e., a unique identifier for the credential in the credential store.
- Obtain the description of all "relevant" established pseudonyms in the credential store by calling `CredentialManager.getPseudonymsWithMetaData(scope`), where scope is an attribute that is specified in the presentation policy and indicates a string to which previously established pseudonyms are associate. If the presentation policy asks to present different pseudonyms with different scopes, this method may be called iteratively for all scope strings.
- Generate the description (i.e., the token without the cryptographic evidence) of all possible presentation tokens that can be created to satisfy the presentation policy using different combinations of credentials and pseudonyms. The list of presentation tokens also includes tokens that establish new pseudonyms (rather than re-using existing pseudonyms) for those pseudonyms where the policy allows it.
- Invoke the IdentitySelection to choose among the different possible presentation tokens by calling `IdentitySelection.selectPresentationTokenDescription(creddescs, pseudonyms, tokendescs, creduids)` which returns the selected presentation token description, the list of credential UIDs to generate it, and a list of metadata for the pseudonyms.

- For all credentials used in the selected token that have Issuer-driven or Verifier-driven revocation restrictions and for which the presentation token description (and hence the presentation policy) explicitly mention the revocation information UID with respect to which the non-revocation status must be shown, call `CredentialManager.hasBeenRevoked(creduid, revparsuid, revokedatts, revinfouid);` for all other revocation restrictions mentioned in the presentation token description, call `CredentialManager.hasBeenRevoked(creduid, revparsuid, revokedatts).` Both of these methods check whether the credential has been revoked and update the non-revocation evidence if necessary, but the first offers the privacy advantage of not "pinging" the Revocation Authority unnecessarily for the latest revocation information. If any of the credentials turns out to be revoked, go back to step 4 with the updated set of possible presentation tokens and let the User/IdentitySelection select a different presentation token.
- Let the evidence generation orchestration generate the cryptographic evidence for the chosen token by calling `EvidenceGenerationOrchestration.createToken(tokendesc, creduids)` which returns the full presentation token, including cryptographic evidence.
- Attach user-defined metadata to all pseudonyms used in the presentation token by calling `CredentialManager.attachMetadataToPseudonym(pseudonym, metadata)` for every pseudonym in the token.
- Return the presentation token.

Alternatively, one could switch the order of steps 4 and 5 so that the `createToken` method first checks whether any credential involved in *any* of the possible presentation tokens has been revoked. This approach has the advantage of never having to ask the User to make a new selection because a credential was revoked, but has the drawback in efficiency and privacy that the Revocation Authorities of *all* possible presentation tokens get "pinged" during presentation, rather than just the Revocation Authorities of the selected presentation token.

`canBeSatisfied(p:PresentationPolicyAlternatives) → boolean`

This method performs steps 1-3 of `PolicyCredentialMatcher.createToken(p)` and returns true if and only if the list of possible presentation tokens is non-empty. Note that this does not include any revocation checks, i.e., the response of this method is more an indication whether a presentation policy can, in general, be fullfilled or not.

`createIssuanceToken(m: IssuanceMessage, atts:Attribute[]) → IssuanceMessage`

This method returns an issuance message with an issuance token that satisfies the given issuance policy and that contains the extended cryptographic evidence, as well as the cryptographic data to support the "carried-over" attributes. It first performs the same credential/pseudonym selection steps as in the presentation scenario to investigate whether the User has the necessary credentials and/or established pseudonyms to satisfy the issuance policy, and in particular the presentation policy part within it. If there are multiple ways in which the policy can be satisfied (e.g., by using different sets of credentials), this method will invoke the IdentitySelection, possibly presented by a user interface to let the user choose her preferred way of generating the presentation token. The steps 1-3, 6-7 are similar or identical to the steps in the `Policy CredentialMatcher.createPresentationToken()` method, see the description above for a detailed description.

- `CredentialManager.getCredentialDescriptions(issuers, credspecs)`
- `CredentialManager.getPseudonymsWithMetaData(scope).`
- Generates all possible issuance token descriptions using the given credentials, pseudonyms, and self-claimed attributes
- The Identity Selection is invoked to select the preferred token description, credentials, pseudonyms and attributes for the issuance token. To this end, the method passes a list of credential descriptions, a list of pseudonyms, a list of the possible issuance token descriptions containing the credential template that was specified in the issuance token, corresponding credential-identifier lists and a list of self-claimed attributes (if given as input) to the Identity Selection. As in the presentation case, the Identity Selection responds with the chosen description and credentials, but here the `IdentitySelection` can also extend or modify the set of self-claimed attributes into `newatts` which will be incorporated in the newly issued credential.
  `IdentititySelection.selectIssuanceTokenDescription(creddescs, pseudonyms, issuancetokendescs, credlist, atts) ⟶ (issuancetokendesc, creduids, pseudometadata, newatts)`
- For all credentials used in the selected token description that have Issuer-driven or Verifier-driven revocation restrictions, the methods
  `CredentialManager.hasBeenRevoked(creduid, revparsuid, revokedatts, revinfouid)`
  and
  `CredentialManager.hasBeenRevoked(creduid, revparsuid, revokedatts)`
  are called to check whether the credential has been revoked, and to update the non-revocation evidence if necessary. If any of the credentials turns out to be revoked notify the User and go back to step 4 to let the User select a different presentation token.
- When the preferred issuance token description was selected, it invokes the `EvidenceGenerationOrchestration` to generate the issuance token. The call also includes the new attributes `newatts` and the context attribute extracted from the IssuanceMessage to allow book-keeping of local status information.
  `EvidenceGenerationOrchestration.createIssuanceToken(issuancetokendesc, creduids, newatts, ctxt)`
- `CredentialManager.attachMetadataToPseudonym(pseudonym, metadata)`
  (for every pseudonym in the token.)
- Return the `IssuanceMessage` that contains the Issuance Token.

5.2.2 Issuance Manager (User)

`issuanceProtocolStep(m:IssuanceMessage, atts:Attribute[]) → IssuanceMessage or CredentialDescription`

This method performs one step in an interactive issuance protocol. On input an incoming issuance message `m` obtained from the Issuer, it either returns the outgoing issuance message that is to be sent back to the Issuer, or returns a description of the newly issued credential at successful completion of the protocol. In the former case, the Context attribute of the outgoing message has the same value as that of the incoming message, allowing the Issuer to link the different messages of this issuance protocol.

The first time this method is called in an issuance protocol instance (i.e., the first time it is called for a new Context attribute in the issuance message `m`), the input parameter `atts` contain attribute-value pairs for self-claimed attributes that are to be included in the issued credential. In all subsequent invocations for the same issuance protocol instance, the input parameter `atts` is simply ignored.

If the incoming issuance message `m` contains an issuance policy, this method will investigate whether the User has the necessary credentials and/or established pseudonyms to create an issuance token that satisfies the issuance policy. If there are multiple ways in which the policy can be satisfied (e.g., by using different sets of credentials), this method will invoke a user interface to let the user choose her preferred way of generating the presentation token.

If the incoming issuance message `m` contains an issuance policy, this method invokes the method `PolicyCredentialMatcher.createIssuanceToken(ip, atts, ctxt )` on the issuance policy `ip` and context attribute `ctxt` from the issuance message and the attribute list.

Otherwise, this method performs one step in an interactive issuance protocol by calling `CryptoEngine.issuanceProtocolStep(m)` and returns the obtained output, which is either an outgoing issuance message or the description of the newly issued credential.

5.2.3 CredentialManager (User)

`getCredentialDescriptions(issuers:anyURI[], credspecs:anyURI[]) → CredentialDescription[]`

This method returns the descriptions of credentials in the credential store with Issuer parameter UID occurring in the input parameter `issuers` and with credential specification UID occurring in `credspecs`. The credential description contains the attributes of the credential as well as possibly a “friendly name” or picture for the credential.

`getCredentialDescription(creduid:anyURI) → CredentialDescription`

This method returns the description of the credential with the given unique identifier `creduid`.

`getCredential(creduid:anyURI) → Credential`

This method returns the full credential (including description, cryptographic metadata, and stored non-revocation evidence) with the given unique identifier.

`getPseudonymWithCryptoData(p:Pseudonym) → PseudonymWithCryptoData`

This method returns the stored pseudonym and attached cryptographic metadata for the given pseudonym `p`.

`getPseudonymsWithMetaData(scope:String) → PseudonymWithMetadata[]`

This method returns a list of pseudonyms and their attached non-cryptographic metadata for the given `scope`.

`storePseudonymWithCryptoData(p:Pseudonym, c:CryptoParams)`

This method stores the given pseudonym together with the cryptographic metadata in the credential store.

`attachMetadataToPseudonym(p:Pseudonym, md:PseudonymMetadata)`

This method attaches the non-cryptographic metadata `md` to the stored pseudonym `p`. The metadata `md` is attached in addition to any metadata that was already attached to the pseudonym.

`hasBeenRevoked(creduid:anyURI, revparsuid:anyURI, revokedatts:anyURI[]) → boolean`

This method returns true if the credential with identifier `creduid` has been revoked by the Revocation Authority with parameters identifier `revparsuid` with respect to the combination of attributes `revokedatts`, and returns false otherwise.

As a side effect, this method fetches the latest revocation information from the Revocation Authority by calling `KeyManager.getRevocationInformation(revparsuid)` and, if necessary, updates the non-revocation evidence of the specified credential with respect to the given Revocation Authority and combination of attributes by calling
`CryptoEngine.updateNonRevocationEvidence(cred, revparsuid, revokedatts)`
and storing the returned credential with the updated non-revocation evidence. If the credential has been revoked, the `CryptoEngine.updateNonRevocationEvidence` method returns an error saying so, and this method returns true. In case the update ended without an error message, this method returns false.

`hasBeenRevoked(creduid:anyURI, revparsuid:anyURI, revokedatts:anyURI[], revinfouid:anyURI) → boolean`

This method returns true if the credential with identifier `creduid` has been revoked by the Revocation Authority with parameters identifier `revparsuid` with respect to the combination of attributes `revokedatts` and with respect to the current revocation information `revinfouid`.

The main difference with the previous method is that this one does not fetch the latest revocation information from the Revocation Authority, but uses the input parameter `revinfouid` instead. It does so by calling
`CryptoEngine.updateNonRevocationEvidence(cred, revparsuid, revokedatts, revinfouid)`
and storing the returned credential with the updated non-revocation evidence. If the credential has been revoked, the `CryptoEngine.updateNonRevocationEvidence` method returns an error saying so, and this method returns true. In case the update ended without an error message, this method returns true.

`updateNonRevocationEvidence()`

This method updates the non-revocation evidence associated to all credentials in the credential store by calling `CryptoEngine.updateNonRevocationEvidence(cred, raparsuid, revokedatts)`for all credentials `cred` in the store and all non-revocation evidences associated to them. The returned credential with updated non-revocation evidence are stored together in the credential store.

`storeCredential(cred: Credential) → anyURI`

This method saves the given credential, consisting of a credential description and the cryptographic data in permanent storage and also enhances the credential description by a unique identifier which is assigned to the credential. The return value is the unique identifier.

`deleteCredential(creduid:anyURI) → boolean`

This method deletes the credential with the given identifier from the credential store. If deleting is not possible (e.g. if the refered credential does not exist) the method returns false, and true otherwise.

`listCredentials()→ anyURI[]`

This method returns an array of the unique identifiers of all stored credentials.

5.2.4 IdentitySelection (User)

```
selectPresentationTokenDescription(creddescs:CredentialDescription[],
nyms:PseudonymWithMetadata[], tokens:PresentationTokenDescription[],
creduids:CredentialUID[][]) → (PresentationTokenDescription,
CredentialUID[], PseudonymMetadata[])
```

This method performs the identity selection, possibly presented by a graphical user interface, allowing the User to choose which combination of credentials and/or pseudonyms, all satisfying the policy she prefers to use. It takes the following input parameters:

- `creddescs`: The list of credential descriptions of all credentials that are used in any of the candidate tokens.
- `nyms`: The list of pseudonyms with their metadata of all pseudonyms that are used in any of the candidate tokens.
- `tokens`: The list of candidate presentation tokens.
- `creduids`: A two-dimensional list specifying for each candidate presentation token which credentials would be used to generate it. Meaning, `creduid[i]` is the list of credential identifiers that are used to generate `token[i]`. The list of credential identifiers is sorted according to the order that they appear in the presentation token.

The method returns the chosen presentation token and the list of identifiers of the credentials used to generate it. It also returns user-defined metadata for each pseudonym in this presentation token, in the order that the pseudonyms occur in the token. Using user-defined meta-data, the User can add free notes or descriptions to the pseudonym to remind her later when re-using the pseudonym. The metadata will be stored with the pseudonym.

```
selectIssuanceTokenDescription(creddescs:CredentialDescription[],
nyms:PseudonymWithMetadata[],
issuancetokendescs:IssuanceTokenDescription[], credlist:CredentialUID[][],
atts: Attribute[]) → (IssuanceTokenDescription, CredentialUID[],
PseudonymMetadata[], Attribute[])
```

This method is an "enhanced" version of the `selectPresentationTokenDescription()` method above. This method presents again an identity selection, possibly being a graphical user interface allowing the User to choose which combination of credentials and/or pseudonyms she prefers to satisfy the policy and which self-claimed attributes she wants to embed in the new credential to be issued. It takes the following input parameters:

- `creddescs`: The list of credential descriptions of all credentials that are used in any of the candidate tokens.
- `nyms`: The list of pseudonyms with their metadata of all pseudonyms that are used in any of the candidate tokens.
- `issuancetokendescs`: The list of candidate issuance tokens. Each token also contains the credential template that describes which attributes from which credentials will be carried over to the newly issued credential.
- `creduids`: A two-dimensional list specifying for each candidate presentation token which credentials would be used to generate it. Meaning, `creduid`[i] is the list of credential identifiers that are used to generate `token`[i]. The list of

credential identifiers is sorted according to the order that they appear in the presentation token.

- `atts`: The list of attribute type-value pairs containing the self-claimed attributes.

In the course of the identity selection, the User can also add new self-claimed attributes (if allowed by the credential template) which will be inserted into the new credential, or modify the suggested values.

The method returns the chosen issuance token, the list of identifiers of the required credentials, metadata for each pseudonym in this token and the list of self-claimed attributes.

5.2.5 EvidenceGenerationOrchestration (User)

```
createPresentationToken(td:PresentationTokenDescription, creds:anyURI[]) →
PresentationToken
```

This method generates the cryptographic evidence for the given presentation token description `td` using the credentials with UIDs mentioned in `creds` (given in the order that the credentials appear in `td`). The output of the method is a presentation token containing both the token description and the evidence.

If the presentation token can be separated into multiple subtokens (e.g., one subtoken using a Privacy-ABC technology and another one using X.509), then this method splits up the token description `td` and the credential identifiers `creds` per subtoken and, for each subtoken, generates the cryptographic evidence by calling `CryptoEngine.createToken (subtokendesc, subcreds).` It then assembles all returned subtokens into a single presentation token.

If there is only a single subtoken, this method simply makes a single call `CryptoEngine. createToken(tokendesc, creds)` and returns the resulting presentation token.

```
createIssuanceToken(itd: IssuanceTokenDescription, creduids:anyURI[],
atts:Attribute[], ctxt:String) →  IssuanceMessage
```

This method orchestrates the generation of the cryptographic evidence for the given issuance token description `itd` using the credentials with UIDs mentioned in `creduids` (given in the order that the credentials appear in `itd`). The output of the method is an issuance message that encapsulates a token containing both the issuance token description and the evidence. Here the evidence can also contain additional cryptographic data which will subsequently be used in the issuance protocol for the carried-over attributes.

If the issuance token can be separated into multiple subtokens (e.g., one subtoken using a Privacy-ABC technology and another one using X.509), then this method splits up the token description and the credential identifiers per subtoken and, for each subtoken, generates the cryptographic evidence by calling
`CryptoEngine.createIssuanceToken(subissuancetokendesc, subcreduids, atts, ctxt).`

This call also includes a Context attribute `ctxt` to allow the *`CryptoEngine`* to bind cryptographic state information of different subtokens to one issuance session.

It finally assembles all returned subtokens into a single Issuance token, which in turn is wrapped into an `IssuanceMessage` with Context `ctxt`.

If there is only a single subtoken, this method simply makes a single call
`CryptoEngine.createIssuanceToken(issuancetokendesc, creduidsatts, ctxt)`

5.2.6 CryptoEngine (User)

`setupUser(klength:integer, muid:anyURI) → anyURI`

> `This method, on input a key length klength and the identifier muid of the cryptographic mechanism to be used, generates a cryptographically strong user secret that is kept in trusted storage. The method returns the identifier to which the user secret is associated.`

`createPresentationToken(td:PresentationTokenDescription, creds:anyURI[]) → PresentationToken`

This method generates the cryptographic evidence for the given presentation token description td using the credentials with UIDs mentioned in `creds` (given in the order that the credentials appear in `td`).

This method also saves newly established pseudonyms including their scope and any cryptographic metadata (to allow later re-authentication under the pseudonym) in permanent storage. The User-generated metadata will be associated to the pseudonym by the `PolicyCredentialMatcher.createToken` method when the token is returned.

The token is created by taking the following steps:

7. Fetch the full credentials and pseudonyms required to generate the token by repetitively calling the methods
   `CredentialManager.getCredential(creduid)`
   `CredentialManager.getPseudonymWithCryptoData(pseudonym).`
8. Fetch the Issuer parameters, Inspector public keys and Revocation Authority parameters needed to generate the token from the KeyManger by invoking the following methods:
   `KeyManager.getIssuerParameters(issuid)`
   `KeyManager.getInspectorPublicKey(ipkuid)`
   `KeyManager.getRevocationParameters(rapuid)`
9. Invoke mechanism-specific cryptographic routines to generate the cryptographic evidence for the token.
10. Store the cryptographic metadata used to generate newly established pseudonyms in the credential store using the method
   `CredentialManager.storePseudonymWithCryptoData(pseudonym, CryptoParams).`
11. Assemble the full (sub)presentation token from the token description and the generated evidence and return the (sub)token.

`createIssuanceToken(itd: IssuanceTokenDescription, creduids: CredentialUID[], atts: Attribute[], ctxt:String) → IssuanceToken`

This method generates the extended cryptographic evidence for the given issuance token description `itd` using the credentials listed in `creduids` (given in the order that the credentials appear in `itd`).

This method also keeps state information (such as the randomness of commitments) that might be required in a later step of the issuance protocol. To be able to identify the information again, the method associates the data to the unique Context attribute `ctxt`.

- As in the normal `createToken` method, it first fetches the full credentials and pseudonyms that are specified in `creduids` and the issuance token description and also obtains all required key material with the help of the *`KeyManager`*. (See the `CryptoEngine.createToken` method for a detailed description)

- The method invokes mechanism-specific cryptographic routines to generate the cryptographic evidence for the issuance token.
- It keeps the state information that might be required in a subsequent step of the issuance protocol in a temporary storage associated to the context.
- It stores the newly generated pseudonyms including their cryptographic metadata using the `CredentialManager` (See the `CryptoEngine.createToken` method for a detailed description)
- If the newly issued credential is subject to Issuer-driven revocation restrictions, then, depending on the revocation mechanism, the *`CryptoEngine`* may have to interact with the Revocation Authority during issuance. If so, then this method prepares a mechanism-specific `RevocationMessag`e *m* and calls `RevocationProxy.processRevocationMessage(m, rapars)`.
- It returns the (sub)issuance token to the `EvidenceGenerationOrchestration`.

`issuanceProtocolStep(m:IssuanceMessage) → IssuanceMessage or CredentialDescription`

On input an incoming issuance message `m`, this method first extracts the context attribute and obtains the cryptographic state information that is stored under the same context value. It then invokes the mechanism-specific cryptographic routines for one step in an interactive issuance protocol.

If the newly issued credential is subject to Issuer-driven revocation, then, depending on the revocation mechanism, this method may interact with the Revocation Authority by calling `RevocationProxy.processRevocationMessage(m, revpars)`.

The method either returns an outgoing issuance message or a description of the newly issued credential to indicate a successful completion of the protocol. In the former case, the method eventually also stores new cryptographic state information associated to the Context attribute, and attaches the Context attribute to the outgoing message. If the invoked cryptographic routines complete the issuance protocol, the method stores the obtained credential with all the cryptographic metadata in the credential store by calling
`CredentialManager.storeCredential(cred: Credential)`
and returns the credential description.

`updateNonRevocationEvidence(cred:Credential, raparsuid:anyURI, revokedatts:anyURI[]) → Credential`

This method updates the non-revocation evidence stored in credential `cred` with respect to Revocation Authority parameters `raparsuid` and with respect to attribute combination `revokedatts`, or tries to create such non-revocation evidence when it does not exist yet. It returns the credential with updated non-revocation evidence.

This method always updates the non-revocation to the most current state. It calls `KeyManager.getCurrentRevocationInformation(raparsuid)` to obtain the most recent revocation information, and possibly calls the `RevocationProxy.processRevocationMessage(m, rapars)` method to interact with the Revocation Authority.

`updateNonRevocationEvidence(cred:Credential, raparsuid:anyURI, revokedatts:anyURI[], revinfouid:anyURI) → Credential`

This method updates the non-revocation evidence stored in credential `cred` with respect to Revocation Authority parameters `raparsuid` and with respect to attribute combination `revokedatts`, or tries to create such non-revocation evidence when it does not exist yet. It returns the credential with updated non-revocation evidence.

Contrary to the previous method, this method updates the non-revocation information so that it can be verified against the given revocation information `revinfouid`, which may not be the latest revocation information for the Revocation Authority parameters. It may call the `RevocationProxy.processRevocationMessage(m, rapars)` method to interact with the Revocation Authority.

5.2.7 KeyManager (all)

`getIssuerParameters(issuid:anyURI) → IssuerParameters`

This method returns the Issuer parameters with the given unique identifier `issuid`, or returns nothing (null) if no such parameters can be obtained in a trusted way.

`getInspectorPublicKey(ipkuid:anyURI) → InspectorPublicKey`

This method returns the Inspector public key with the given unique identifier `ipkuid`, or returns nothing (null) if no such public key can be obtained in a trusted way.

`getCredentialSpecification(credspecuid:anyURI) → CredentialSpecification`

This method returns the Credential Specification with the given unique identifier `credspecuid`, or returns nothing (null) if no specification can be obtained in a trusted way.

`getRevocationAuthorityParameters(rapuid:anyURI) → RevocationAuthorityParameters`

This method returns the Revocation Authority parameters with the given unique identifier `rapuid`, or returns nothing (null) if no such parameters can be obtained in a trusted way.

`getCurrentRevocationInformation(rapuid:anyURI) → RevocationInformation`

This method returns the current revocation information for the given Revocation Authority parameters `rapuid`. The *KeyManager* may have the current revocation information cached, but if not, it looks up the appropriate endpoint in the Revocation Authority parameters and fetches the latest revocation information from the Revocation Authority directly.

`getRevocationInformation(rapuid:anyURI, revinfouid:anyURI) → RevocationInformation`

This method returns the revocation information with identifier `revinfouid` for the given Revocation Authority parameters `rapuid`. The *`KeyManager`* may have the requested revocation information cached, but if not, it looks up the appropriate endpoint in the Revocation Authority parameters and fetches the requested revocation information from the Revocation Authority directly.

The requested revocation information `revinfouid` may not be the latest revocation information. Note the difference with the `getCurrentRevocationInformation` method, which always returns the latest revocation information.

5.2.8 RevocationProxy (User, Verifier, Issuer)

`processRevocationMessage(m:RevocationMessage, revpars:RevocationAuthorityParameters) → RevocationMessage`

This method carries out one step in a possibly interactive protocol with a Revocation Authority. Depending on the revocation mechanism, interaction with the Revocation Authority may be needed during credential issuance, during creation or verification of a presentation token, or during the separately triggered creation or update of non-revocation evidence.

This method will be called by the *`CryptoEngine`* of the User, Verifier, or Issuer whenever it needs to interact with the Revocation Authority. The method acts as a proxy for the communication with the Revocation Authority: in the parameters `revpars` it looks up the appropriate endpoint to contact the Revocation Authority, sends the outgoing revocation message `m`, and returns the response received from the Revocation Authority to the local *`CryptoEngine`*.

The returned message will have the same Context attribute as the outgoing message `m`, so that the different messages in a protocol execution can be linked.

5.2.9 PolicyTokenMatcher (Verifier)

```
verifyTokenAgainstPolicy(p:PresentationPolicyAlternatives,
t:PresentationToken, store:boolean) → PresentationTokenDescription or
string[]
```

This method checks that the given token `t` satisfies one of the given policies contained in `p` and that the cryptographic evidence in token t is valid. If the input parameter store is set to true, the token is saved in permanent storage and assigned a unique token identifier by means of which it can later be retrieved. If all checks succeed, the method returns the verified presentation token description. This is essentially the token stripped from all cryptographic evidence. If `store` is true, the returned token includes the unique token identifier generated by the TokenManager. If any of the checks fail, this method returns a list of error messages giving more detail about the reason for failure.

To verify a token, this method takes the following steps:

- It first separates the token description from the cryptographic evidence and checks that the token description satisfies a presentation policy by calling `PolicyTokenMatcher.matchPresentationTokenDescriptionAgainstPolicy(p, tdesc)`
- It then passes the presentation token to the `EvidenceVerificationOrchestration.verifyToken(t)` method to dissect the token in subtokens and verify the cryptographic evidence.
- If store is true, it saves the token in permanent storage by calling `TokenManager.storeToken(t).`
  The storeToken method returns a unique identifier for the stored token.
- It returns the description of presentation token `t`, possibly enhanced with the unique identifier obtained in the previous step.

```
matchPresentationTokenDescriptionAgainstPolicy(p:PresentationPolicyAlternat
ives, td:PresentationTokenDescription) → boolean or string[]
```

This method decides whether the given presentation token description `td` satisfies one of the presentation policies contained in `p`. In order to do so, it may call out to `TokenManager.isEstablishedPseudonym(pseudonym)` to check whether a pseudonym contained in `td` is an established pseudonym.

5.2.10 TokenManager (Verifier)

`isEstablishedPseudonym(p:Pseudonym) → boolean`

> This method checks whether the given pseudonym `p` is an established pseudonym, i.e., whether the pseudonym occurs in any stored presentation tokens.

`storeToken(t:PresentationToken) → anyURI`

> This method saves the given presentation token `t` in permanent storage and assigns a unique identifier to the token by means of which it can later be retrieved. The return value is the unique identifier.

`getToken(tokenuid:anyURI) → PresentationToken`

> This method looks up a previously verified presentation token by the unique identifier `tokenuid`.

`deleteToken(tokenuid:anyURI) → boolean`

> This method deletes the previously verified presentation token referenced by the unique identifier tokenuid. It returns true in case of successful deletion, and false otherwise.

5.2.11 EvidenceVerificationOrchestration (Verifier)

`verifyToken(t:PresentationToken) → boolean or string[]`

> This method dissects the given presentation token `t` in `subtokens`, separating those `subtokens` based on Privacy-ABCs from those based on other ABC technologies such as X.509. For each of these `sub-tokens`, it invokes `CryptoEngine.verifyToken(subtoken)` to verify the cryptographic evidence of the `subtoken`. If all `subtokens` are deemed valid, this method returns true, otherwise it returns a list of error messages.

5.2.12 CryptoEngine (Verifier)

`verifyToken(t:PresentationToken) → boolean or string[]`

> This method verifies that the cryptographic evidence in the given presentation token t supports the description of `t`. If the evidence is deemed valid, this method returns `true`, and a list of error messages otherwise.
>
> In order to verify the token, this method may call upon the *`KeyManager`* to obtain Issuer parameters, Inspector public keys, Revocation Authority parameters, and the current revocation information by invoking the methods
> `KeyManager.getIssuerParameters(issuid)`,
> `KeyManager.getInspectorPublicKey(ipkuid)`,
> `KeyManager.getRevocationAuthorityParameters(rapuid)`,
> and
> `KeyManager.getCurrentRevocationInformation(rapuid)`.

5.2.13 IssuanceManager (Issuer)

`initIssuanceProtocol(ip:IssuancePolicy, atts:Attribute[]) →`
`(IssuanceMessage, boolean)`

> This method is invoked by the Issuer to initiate an issuance protocol based on the given issuance policy `ip` and the attribute values `atts` to be embedded in the new credential. It returns an Issuance Message that is to be sent to the User and fed to the

`IssuanceManager.issuanceProtocolStep` method on the User's side. The Issuance Message contains a Context attribute that will be the same for all message exchanges in this issuance protocol, to facilitate linking the different flows of the protocol. It also outputs a boolean indicating whether this is the last flow of the issuance protocol.

In case of an issuance "from scratch", i.e., for which the User does not have to prove ownership of existing credentials or established pseudonyms, the given issuance policy `ip` merely specifies the credential specification and the issuer parameters for the credential to be issued. In this case, the returned issuance message is the first message in the actual cryptographic issuance protocol. This method will then directly invoke
`CryptoEngine.initIssuanceProtocol(ip, null, atts, ctxt)`
on input the issuance policy (containing only the credential specification UID), the list of known attributes and a freshly generated Context value `ctxt` that is used for local "bookkeeping" of the cryptographic state and for tying the issuance protocol messages together.

The returned issuance message will contain the fresh Issuer Context attribute that links the different messages of this issuance protocol together.

In case of an "advanced" issuance, i.e., where the User has to prove ownership of existing credentials or pseudonyms to carry over attributes, a user secret, or a device secret, the returned Issuance Message is simply a wrapper around the issuance policy `ip` with a fresh Context attribute.

`issuanceProtocolStep(m:IssuanceMessage) → (IssuanceMessage, boolean)`

This method performs one step in an interactive issuance protocol. If the incoming issuance message `m` does not contain an issuance token received from a User, it calls the *`CryptoEngine`* on `m`:
`CryptoEngine.issuanceProtocolStep(m)`
The method then returns the output of the *`CryptoEngine`*, which can be either an outgoing issuance message, or a description of the newly issued credential at successful completion of the protocol. In the former case, the Context attribute of the outgoing message has the same value as that of the incoming message, allowing to link the different messages of this issuance protocol.

If the incoming issuance message `m` does contain an issuance token `it`, then this method generates the first message of the actual issuance protocol in an "advanced" issuance, i.e., where the User has to prove ownership of existing credentials or pseudonyms to carry over attributes, a user secret, or a device secret. The issuance policy `ip` and the list of attribute type-value pairs `atts` was given as input to the
`IssuanceManager.initIssuanceProtocol` when the Context for this issuance protocol instance was defined.

This method first verifies the validity of the Issuance Token by calling *`PolicyTokenMatcher`* on input Issuance Policy and Issuance Token.
`PolicyTokenMatcher.verifyIssuanceTokenAgainstPolicy(ip, it, store)`

The *`PolicyTokenMatcher`* verifies in two steps whether the issuance token description satisfies the policy and whether the cryptographic evidence supports the token description. If both checks succeed, it obtains the verified presentation token description (including a unique token identifier in case `store` was set to true).

If the verification of the Issuance Token was successful, it subsequently invokes the *`CryptoEngine`* to perform the first round of the actual issuance protocol. To this end, the method passes the issuance policy, the issuance token (for the carried-over attributes), the

attributes chosen by the Issuer and the context attribute from the issuance message `m` to the *`CryptoEngine`*.
`CryptoEngine.initIssuanceProtocol(ip, it, atts, ctxt)`

The output of this method is an `Issuance Message` which is the first move in the actual issuance protocol. The returned issuance message will contain the same Context identifier `ctxt` that was included in the issuance message `m` to link the different messages of this issuance protocol.

In both cases, also a boolean is returned which indicates whether the IssuanceMessage is the final flow in the issuance protocol.

5.2.14 PolicyTokenMatcher (Issuer)

```
verifyIssuanceTokenAgainstPolicy(ip:IssuancePolicy, it:IssuanceToken,
store:boolean) → IssuanceTokenDescription or string[]
```

Similarly to the `verifyTokenAgainstPolicy` method, this method checks that the given token satisfies the given policy and that the cryptographic evidence in the token is valid. As this method deals with issuance policies (consisting of a presentation policy and credential template) and issuance tokens, it also checks whether the requirements concerning carried-over or (jointly-)random attributes are fulfilled.

If the input parameter `store` is set to true, the token is saved in permanent storage, assigned to a unique token identifier. If all checks succeed, the method returns the verified issuance token description. This is essentially the token `it` stripped from all cryptographic evidence. If `store` is true, the returned token includes the unique token identifier. If any of the checks fail, this method returns a list of error messages giving more detail about the reason for failure.

To verify a token, this method takes the following steps:

- It first separates the token description from the cryptographic evidence and checks that the issuance token description satisfies the presentation policy and credential template contained in the issuance policy by calling `PolicyTokenMatcher.matchIssuanceTokenDescriptionAgainstPolicy(ip, itdesc)`
- It then passes the issuance token to the `EvidenceVerificationOrchestration.verifyIssuanceToken(ip, it)` method to dissect the token in subtokens and verify the cryptographic evidence. This method also verifies if the carried-over or (jointly-)random attribute requirements as specified in the credential template are fulfilled. The method will respond with true in case the checks succeed, and with an error message otherwise.
- If `store` is true, it saves the token in permanent storage by calling `TokenManager.storeToken(it).`
  The `storeToken` method returns a unique identifier for the stored token.
- It returns the description of issuance token, possibly enhanced with the unique identifier obtained in the previous step.

```
matchIssuanceTokenDescriptionAgainstPolicy(ip:IssuancePolicy,
itdesc:IssuanceTokenDescription)
```

This method verifies that the presentation token description `itdesc` satisfies the issuance policy `ip`. It performs all the checks that are performed by the

`PolicyTokenMatcher.matchPresentationTokenDescription` method, and moreover checks that the presentation token contains (only) those self-claimed attributes that are required by the policy.

5.2.15 EvidenceVerificationOrchestration (Issuer)

`verifyIssuanceToken(it:IssuanceToken) → boolean or string[]`

This method dissects the given issuance token in subtokens, separating those subtokens based on Privacy-ABCs from those based on other ABC technologies such as X.509. For each of these subtokens, it invokes

`CryptoEngine.verifyIssuanceToken(subtoken)`

to verify the cryptographic evidence of the subtoken. If all subtokens are deemed valid, this method returns true, otherwise it returns a list of error messages.

5.2.16 CryptoEngine (Issuer)

```
setupSystemParameters(keylength:integer, mechanism:anyURI) →
SystemParameters
```

This method generates a fresh set of system parameters for the given key length and cryptographic mechanism. Issuers can generate their own system parameters, but can also reuse system parameters generated by a different entity.

```
setupIssuerParameters(credspec:CredentialSpecification,
syspars:SystemParameters, uid:anyURI, hash:anyURI, revparsuid:anyURI) →
IssuerParameters
```

This method generates a fresh issuance key and the corresponding Issuer parameters. The issuance key is stored in trusted storage, the Issuer parameters are returned as output of the method. The input to this method specify the credential specification credspec of the credentials that will be issued with these parameters, the system parameters syspars, the unique identifier uid of the generated parameters, the hash algorithm identifier hash, and, optionally, the parameters identifier for any Issuer-driven Revocation Authority.

`verifyIssuanceToken(it:IssuanceToken) → boolean or string[]`

This method verifies that the cryptographic evidence in the given issuance token supports the description of the token. Here the description consists of a presentation token description and a credential template, i.e., this method will particularly check whether the requirements concerning the carried-over and (jointly-)random attributes are met.

In order to verify the token, this method may call upon the `KeyManager` to obtain Issuer parameters, Inspector public keys, Revocation Authority parameters, and the current revocation information.

If the evidence is deemed valid, it returns true and an error description otherwise.

```
initIssuanceProtocol(ip:IssuancePolicy, it:IssuanceToken, atts:Attribute[],
ctxt:String) → (IssuanceMessage, boolean)
```

This method is invoked on an issuance policy (possibly containing only a credential specification UID in case of an issuance “from scratch”), an issuance token (possibly being "empty", i.e. `it=null`), known attributes `atts` and a context string. It invokes the mechanism-specific cryptographic routines for the first step in an interactive issuance protocol and stores its cryptographic state in a temporary storage associated to the context. It finally returns an outgoing issuance message with the same context attribute `ctxt`, plus a boolean indicating whether this is the last flow of the issuance protocol.

If the credential to be issued is subject to Issuer-driven revocation, then, depending on the revocation mechanism, the `CryptoEngine` may have to interact with the Revocation Authority. If so, then it prepares a mechanism-specific Revocation Message `m` and calls `RevocationProxy.processRevocationMessage(m, revpars)`.

`issuanceProtocolStep(m: IssuanceMessage) → (IssuanceMessage, boolean)`

On input an incoming issuance message `m`, this method first extracts the context attribute and obtains the cryptographic state information that is stored under the same context value. It then invokes the mechanism-specific cryptographic routines for one step in an interactive issuance protocol and returns an outgoing issuance message. The method eventually also stores new cryptographic state information associated to the context attribute, and attaches the context attribute to the outgoing message. The returned boolean indicates whether this is the last flow of the issuance protocol. If so, the method deletes all temporary state information.

If the credential to be issued is subject to Issuer-driven revocation, then, depending on the revocation mechanism, the `CryptoEngine` may have to interact with the Revocation Authority. If so, then it prepares a mechanism-specific Revocation Message `m` and calls `RevocationProxy.processRevocationMessage(m, revpars)`.

5.2.17 CryptoEngine (Inspector)

`setupInspectorPublicKey(klength:integer, muid:anyURI, uid:anyURI) → InspectorPublicKey`

This method generates a fresh decryption key and corresponding encryption key for the given key length and cryptographic mechanism. It returns the inspector public key with the given identifier `uid` and keeps the corresponding secret key associated to that uid in trusted storage.

`inspect(t:PresentationToken) → Attribute[]`

This method takes as input a presentation token with inspectable attributes and returns the decrypted attribute type-value pairs for which the Inspector has the inspection secret key. To this end it also fetches the secret descryption key associated with the identifier that is specified in the token from the trusted storage.

5.2.18 CryptoEngine (RevocationAuthority)

`setupRevocationAuthorityParameters(klength:integer, muid:anyURI, uid:anyURI) → RevocationAuthorityParameters`

For a given key length and revocation mechanism, this method generates a fresh secret key for the Revocation Authority and corresponding public Revocation Authority parameters, as well as the initial revocation information. Its keeps the secret key in trusted storage and returns the public key and parameters.

`revoke(rparsuid:anyURI, atts:Attribute[]) → RevocationInformation`

This method revokes the attribute values specified by the input parameter `atts` with respect to the revocation parameters specified by their unique identifier `rparsuid, and using the secret key from the trusted storage corresponding the given identifier.`

`processRevocationMessage(m:RevocationMessage) → (RevocationMessage, boolean)`

This method carries out one step in a possibly interactive protocol with a User or an Issuer during which the User obtains or updates her non-revocation evidence. Depending on the revocation mechanism, such protocols may be part of the issuance of a credential, the creation of a presentation token, or of an independent update of the non-revocation evidence.

The method takes in incoming revocation message `m` and returns an outgoing revocation message that is to be returned as a response to the caller. The outgoing message will have the same Context attribute as the incoming message, so that the different messages in a protocol execution can be linked. The method also returns a boolean to indicate whether this is the last message in the flow. If so, any state information kept for this context can be safely removed.

5.3 Internal Data Formats

The data formats described in this section are exchanged internally between different components of the ABCE layer and of the *`CryptoEngine`*, but are not exchanged "on the wire" between different participants. Since the components may pass information to each other in a native object structure rather than in XML, we do not specify concrete XML schemas here, but rather describe informally what information the data formats must contain.

`PresentationTokenDescription`

This data format contains the non-cryptographic part of a presentation token, i.e., the child element of a `PresentationToken (as defined in Chapter 4)` element that is a sibling of `CryptoEvidence`. It may also contain a unique identifier for the presentation token, in particular, to identify the corresponding full presentation token (with cryptographic evidence) in the Verifier's store when returned by the `PolicyTokenMatcher.verifyTokenAgainstPolicy` method.

`Credential`

This data format contains all cryptographic and non-cryptographic data of a stored credential. It contains all information contained in a credential, including all cryptographic data needed to create a presentation token. Moreover, it contains the issuer parameters, the credential specification, the unique identifier of the credential in the store, the attribute values, as well as all non-revocation evidences (issuer-driven and verifier-driven) for this credential. It may also contain a "friendly name" or any other metadata chosen by the User.

For security reasons, the information passed here may not actually contain all cryptographic secrets needed to create presentation tokens, but merely contain references to it, or offer methods to create presentation tokens using them.

`CredentialDescription`

This data format contains all non-cryptographic information about a Credential. It includes the issuer parameters, the credential specification, the unique identifier of the credential in the store, the attribute values, as well as (non-cryptographic) descriptions of the non-revocation evidences stored for this credential. It also contains usage limitation, if any (e.g., number of usages left). It may also contain a "friendly name" or any other metadata chosen by the User.

`Pseudonym(MetaData/CryptoData)`

This data format can contain cryptographic metadata (CryptoData), allowing the User to re-authenticate under a pseudonym, and/or non-cryptographic metadata (MetaData), allowing the User to attach information for her own reference to the pseudonym. For example, the User could store with which Verifier this pseudonym was established or attach a User-defined "friendly name" for the pseudonym.

6 Legal Considerations

With a strong focus on European data protection law, this chapter gives an overview on legal considerations of deploying Privacy-ABCs. It describes how the benefits of Privacy-ABCs relate to privacy requirements from the legal point of view. The potential impact and contribution of Privacy-ABCs for data protection is described in Section 6.1, followed by a basic overview of typical legal relations between entities deploying Privacy-ABCs (Section 6.2).

The legal analysis and descriptions must necessarily remain on an abstract level. This deliverable is dedicated to the description of the ABC4Trust architecture on a high level and therefore does not provide details of implementations or the planned pilots. Details of the project's pilots will be content of future deliverables. Such details of the use cases highly influence the legal evaluation. The potential use cases and possibilities for deploying Privacy-ABCs differ widely in respect to the business objectives, purposes of processing or the underlying factual relation between the parties. These factors influence the legal relations between the parties. More detailed legal considerations which will be based on the specific use case description of the two ABC4Trust pilots will be part of the upcoming documents dedicated to these pilots.

6.1 Contribution to Privacy and Data Protection

Systems deploying information and communications technology (ICT) should consider privacy and requirements for legal compliance right from the start. An often encountered challenge in practice is the excessive collection and processing of personal data which is not necessary for the given purpose. The problem at hand and the potential solution are closely bound to the privacy principle that only necessary processing of personal data is permitted (principle of necessary processing). This principle can be located in various articles of Directive 95/46/EC.[4] Privacy-ABCs address this problem by allowing selective disclosure. Deploying the principle of necessary processing in technology is complicated as the central aspect to measure and evaluate the necessity against – namely the purpose of the processing – cannot be quantified nor does it seem easily possible to provide a sufficient taxonomy or partonomy describing purposes [HoSch11]. Therefore enforcing this principle currently necessarily requires human interaction to evaluate the necessity.

A noteworthy approach to actually enforce the principle of necessary processing has been undertaken by the German legislator introducing the German eID in 2010.[5] It combines a technical approach with legal measures: To enforce the principle or necessary processing, the German legislator permits service providers to only collect those pieces of personal data from the eID card which are necessary for a previously specified purpose.[6] Only parties that have successfully argued and documented the necessity of the requested categories of personal data towards a dedicated public authority are granted access to the protected content of a German eID card. An analysis of which categories of personal data can be considered necessary for a service provider in a series of typical use cases can be found in [Zwi11].[7] Besides allowing and enforcing selective disclosure of only necessary attributes, the German

4 See in inter alia the articles permitting the processing of data in the absence of an informed consent or limiting such processing to the extent necessary for the stipulated purpose such as Art. 5 para. 1 (b); Art. 7 (b)-(f); Art. 8 para. 2 (b)-(c) of Directive 95/46/EC.

5 An overview of selected jurisdictions within the European eID landscape has been provided by a dedicated issue of the IDIS journal, online: http://www.springerlink.com/content/1876-0678/3/1/.

6 § 21 Sec. 2 Nr. 3 Personalausweisgesetz (German Law on Identity Cards).

7 Note that further data minimisation can be achieved with the appropriate infrastructure supporting Privacy-ABCs as it is explained in the following.

eID allows for pseudonymous age verification and a proof of the place of residence without the need of providing further identifying information [BT08]. As a variety of services only require this type of authentication, this feature of the German eID card makes the processing of further personal data unnecessary.

A potential future deployment of Privacy-ABCs in eID schemas would allow going beyond the existing privacy-preserving capabilities of the German model. Based on several properties Privacy-ABCs bear a high potential to challenge what must have been considered as necessary processing of personal data in the past. If deployed broadly, Privacy-ABCs would allow a revision of the understanding of necessary processing and require reassessment of existing systems. So far three properties of Privacy-ABCs have been identified that directly or indirectly influence the necessity:

First, Privacy-ABCs grant a secure and trustworthy verification of individual attributes out of a larger credential without giving away the whole set of personal data (selective disclosure). Also, the attribute types available for selective disclosure are not limited to the small set that can be vouched for by a public issuer of eIDs (name, nationality, or place and date of birth). Rather, any type of attribute value that can be attested by the Issuer may be content of a credential. Credentials from different authorative sources, meaning entities that can actually vouch for the attribute attested, can be combined. Based on the new possibilities, established methods to process personal data need to be reassessed.

A second challenge for the principle of necessary processing being addressed by Privacy-ABCs is the need of many data controllers to collect personal data just for a more or less likely event that the information is needed at a later stage. This could inter alia be the case for customer-caused non-payment, other kinds of breaches of contract or criminal investigations. The inspection mechanism provided by Privacy-ABCs allows revealing the identity under predefined conditions if this has been chosen by the parties at the time of authentication. For this, the required attribute values are provided and stored in an encrypted format as part of the presentation token (for details see Section 2.6). Unless the previously defined condition is fulfilled, the Verifier cannot access the personal data. The identifying information can be revealed by the Inspector if necessary and if the predefined conditions are met. Inspection therefore allows abstaining from the collection of personal data in clear text for all those business cases which only require identifying and contacting the User under certain conditions.

Third, unlinkability is a key feature of Privacy-ABCs. It allows the enforcement of purpose binding. While the principle of necessary processing is focused on which categories of personal data can be considered required for a purpose and limits the duration of the processing, the principle of purpose binding limits the purposes for which personal data may be processed. It derives from the requirement that personal data must be "collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes", Art. 6 para. 1 (b) of Directive 95/46/EC. Transactions done with Privacy-ABCs cannot be linked to the User or to each other unless attribute values (or other data outside the scope of the ABCs, e.g. from the User's browser in a web transaction) allow for linking and unless it has been explicitly specified otherwise (see Section 2.2), thus preventing tracking and profiling of the Users by Verifiers. This unlinkability of presentation tokens also enforces that the processing of personal data needs to be done separately for each purpose and transaction. Together with the already mentioned possibility of selective disclosure it ensures that only personal data necessary for the respective transaction is retained.

Privacy-ABCs and other privacy-enhancing technologies (PETs) could trigger such a reassessment of the necessity of the processing. However, this requires that it can be reasonably expected from data controllers to deploy these technologies. A respective legal obligation that would clearly state such an obligation for data controllers does not exist. But Art. 17 para. 1 of Directive 95/46/EC requires that data controllers shall implement the "appropriate" measures to protect personal data. This requirement is further limited by the statement that regard should be held to the state of the art and the costs of implementation. Other legal text relating to the deployment of PETs are relatively non-committal as well. So far it has remained rather unclear in legal practice which requirements must be fulfilled to accept an emerging technology as both state of the art and reasonable in price. ISO standardisation of the underlying architecture and provision of an interoperability layer for different cryptographic approaches could be considered a good indication that a technology has reached the state of the art. The costs, being the second condition, must be seen in relation to the nature of the data processed and the total costs of data processing and comprise not only the license fees, but also the cost of setting up and operating the system. Typically costs are dramatically reduced once such a system has reached a certain critical mass in usage and is deployed broadly. This chicken-and-egg problem shared by many PETs could be overcome by national governments or on European level with a legislative initiative enforcing the deployment of PETs or by introducing such technologies on a large scale. For Privacy-ABCs an introduction as a feature of a potential future European ID card could be imagined ensuring broad deployment.

6.2 Basic legal relations

To assess the legal obligations and rights of the parties involved the legal relations need to be identified. Within the basic setup for deploying Privacy-ABCs, several relations exist. Within this document only a first overview is provided based on a simplified system. In general, these legal relations are similar or equal to those between parties deploying classic authentication and identifications schemas. Figure 6.1 illustrates this basic relation. Optional elements are marked with parentheses.

In the description, the usual approach[8] to simplify complex legal multi-party relations into bilateral ones will be followed.

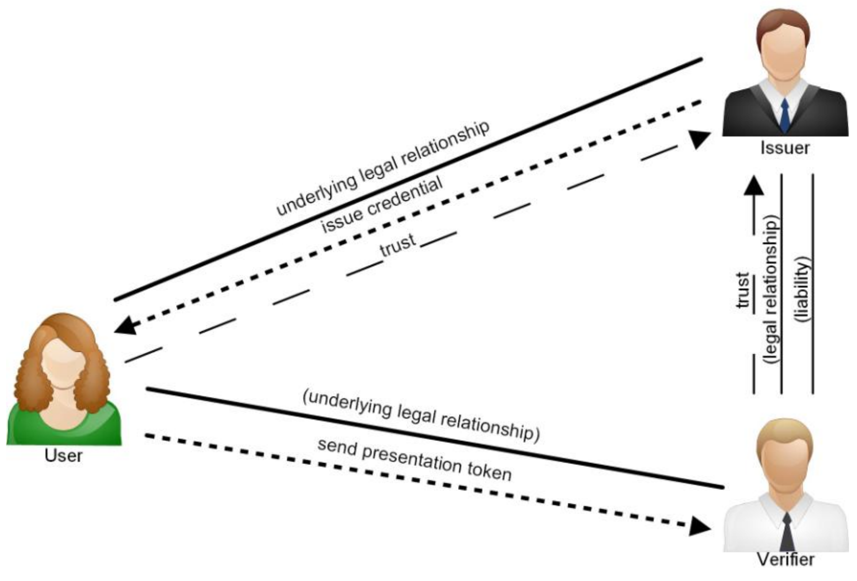

Figure 6.1 - Basic legal relationships

In the basic system of relations within ABCs, three main parties with possible legal relations exist: Issuer, User and Verifier. The Issuer generates and provides credentials containing attributes to the

[8] This is the case as also only two parties can become involved as plaintiff and defendant in a lawsuit at court. Third parties may be involved with special roles supporting one of the parties if the outcome of the disputes directly relates to their own legal positions.

User. The User collects the credentials while interacting with the Issuer, enabling her to provide proof of certain attributes towards the Verifier. The Verifier cryptographically verifies the information in the credential, and checks whether the attribute values provided with the presentation token satisfy the presentation policy. The Verifier is usually holding a resource, is offering a service or is gatekeeper for those, and grants or denies access according to whether a presentation token is provided that satisfies the presentation policy.

The following subsections will describe the legal relations existing between

- User and Issuer (6.2.1),
- User and Verifier (6.2.2), and
- Verifier and Issuer (6.2.3).

6.2.1 Legal relation between User and Issuer

There may be a large variety of underlying legal relations between User and Issuer. In general, the User has some factual relationship with the Issuer, which causes the Issuer to generate and provide credentials containing attributes to the User. Any other detail of the underlying relationship and its legal qualification depend on the individual use case. In commercial relations such as the one between a certification authority and its clients, a specific contract for provision of services can be expected, which also regulates the parties' obligations. Other types of relations such as employment or membership (e.g. clubs, universities, municipalities) may be founded on contractual or quasi-contractual basis. The relation between national states and their citizens, as would be relevant for the issuance of national eIDs, is usually governed by public law.

The User needs to trust the Issuer that personal data will be processed and stored in compliance with privacy requirements. Indirectly, it is also important for the User that the Issuer is considered trustworthy by potential Verifiers and that they accept such credentials. Credentials of an non-trustworthy Issuer will be of little practical value for the User.

<u>Data protection</u>: In the area of data protection law, the underlying factual or contractual relation influences such central aspects as the purpose and the necessity of the processing. As a general rule, the Issuer can be considered to be a data controller in the sense of Directive 95/46/EC and thus is bound to the resulting obligations. The User will likely be the data subject with the according data subject's rights. Exceptions to this rule apply in case of purely personal or household activities on the side of the Issuer or if the data processed are not personal data but anonymous information. In any case, the collection of personal data is limited to the types of personal data and the duration necessary for the purpose. For instance if the Issuer accepts liability towards Verifiers, the Issuer should be allowed to keep a record for which attributes a credential has been issued. However, as with Privacy-ABCs credentials are unlinkable with presentation tokens there is no need to retain copies of the credentials themselves.

<u>Liability</u>: The underlying legal relation between Issuer and User characterises the parties' rights and obligations and potential liability for any violation. Liability of the User is conceivable if she intentionally provides false data to be included in the credential and this actually causes damage for the Issuer. Also not triggering the revocation process in time if a security violation becomes known to the User or using the credential for other than the purposes intended with the issuance might cause indemnifiable damage. If the Issuer has promised performance, she might be held liable for damages incurred due to non-performance. Such a non-performance can be any failure by the Issuer to perform any of her obligations in relation to the issuing process. This includes defective performance, e.g. by

providing a credential with false or missing attributes, not delivering at all, or late issuance of a credential.[9]

6.2.2 Legal relation between User and Verifier

The legal relation between User and Verifier may vary in a range as broad as the possible business cases. The underlying legal relation will most likely be a contract (sale of goods or provision of services, etc.), but may also comprise employment (access management for employees), membership (university, clubs), or contracts with a third party (a student who is member of the university is allowed to access web resources the university subscribed to). Depending on the applicable law also a pure courtesy or favours such as providing information on a privately run website might be treated as a contract-like relation. While these are not likely to create fulfilment obligations, due care may be owed by the parties.

Regarding the necessary trust between User and Verifier, Privacy-ABCs provide a platform for transactions not requiring that the parties trust each other regarding the process of authentication or identification. In particular use cases that only require authentication by specific non-identifying attributes such as age or membership, the parties do not even need to know each other. However, the practical value might be limited if other parts of the transaction (fulfilment by mailing a parcel or payment) require an identification of the parties. To uphold this benefit for data protection gained by Privacy-ABCs, the availability of payment methods that allow anonymity towards the Verifier are necessary.

Data protection: Data protection law requires that certain information about the processing is given to the user enabling her to freely decide whether to agree to the processing or to object, and to enabling her to withdraw consent regarding past processing. To comply with these transparency requirements,[10] the Verifier publishes a privacy policy stating the types of personal data processed and the relevant purposes. In addition a presentation policy describing the content of an acceptable presentation token is necessary. The User then produces a presentation token with the relevant personal data satisfying the presentation policy. Further legal details highly depend on the use case. In most standard use cases where the processing of personal data is a side effect of other legitimate purposes, the law provides a legal ground for the processing.[11] If the law does not provide such a legal ground, the processing of personal data needs to be based on free and informed consent of the data subject. Only in case of an anonymous authentication allowing neither current nor future linking to a person, a legal basis for processing is not required as the data are not personal data in the sense of Art. 2 (a) of Directive 95/46/EC.

Liability: Between Verifier and User liability may result from all type of faults to comply with obligations resulting from the underlying legal relation. However, as such obligations heavily depend on the use case (e.g. not paying, delivering wrong or faulty goods), they are not part of this generic analysis. Specific liability risks related to the process of authentication and identification in general or the use of Privacy-ABCs cannot be seen. In particular, fraudulent conduct such as impersonating

[9] The definition of non-performance is based on Art. 7.1.1. of the UNIDRIOT Principles of International Commercial Contracts 2010. These are based on long-standing efforts in comparative private law reflecting the internationally acceptable baseline for regulating private law relations in commercial trade. Available online: http://www.unidroit.org/english/principles/contracts/main.htm

[10] See inter alia the right of access encoded in Art. 12 of Directive 95/46/EC, or that collection is allowed for clearly specified purposes only, Art. 6 para. 1 (b) of Directive 95/46/EC, or the need to duly inform the data subject about the intended processing to ensure a sufficiently informed (unambiguous) consent according to Art. 7 (a) of Directive 95/46/EC.

[11] See e.g. Art. 7 para. 2 of Directive 95/46/EC permitting inter alia processing for the purpose of performing a contract, or for compliance with legal obligations.

someone else to avoid payment is not assessed differently depending on the type of authentication method chosen for the deceiving action.

6.2.3 Legal relation between Verifier and Issuer

The relation between Verifier and Issuer is primarily based on a trust. The Verifier relies on the Issuer to have provided correct information for the credential. In the basic system with three parties, no legal relationship between these roles is foreseen or necessary. But Verifier and Issuer may also be closely related to each other, e.g. a Verifier may pay the Issuer for the service or the Issuer may be organisational part or subsidiary of the Verifier. This could in particular be the case if Privacy-ABCs are used to replace current forms of authentication such as username and password. Deploying Privacy-ABCs would then still have privacy-enhancing results as an internal anti-profiling measure, a means to enforce the principle of necessary processing and a way to segregate personal data by underlying purposes.

The classification of the legal relation may vary within a wide range. An analysis of the interest in issuing credentials and who is actually paying for the service provided by the Issuer can give some indication of the type of the legal relation. In this respect the Issuer may act on an own interest such as a university or school regarding the pupil or student attribute. In this case a legal relation with Verifiers relying on such a credential such as a cinema offering student discounts may be completely absent. It is further imaginable that the Issuer is a public entity such as the many issuers of national eIDs which act on behalf of national governments. In this case the relation of the Verifier and the Issuer will likely be governed by public law. The User may pay for the service to obtain a credential, e.g. as is current practice with certification authorities issuing certificates for digital signatures. Finally, the Verifier may have an interest to pay the Issuer or even take the role of an Issuer by running a respective service, e.g. as a company issuing credentials for employees to be used internally for access limitation.

Therefore assessing the legal relations between Issuer and Verifier should start by investigating how and by whom the Issuer is financed as this is likely to provide an indication of the underlying legal relation.

Data protection: If personal data are processed, usually both – Issuer and Verifier – can be considered data controllers. Generally any transfer of personal data between different legal entities requires a legal ground. If Verifier and Issuer are part of the same legal entity, a legal ground for the use of personal data is required as well. If the two are different legal entities, the Issuer could act as a data processor according to Art. 16 and Art. 17 of Directive 95/46/EC.

Collaborations between Issuer and Verifier after the issuance do not pose problems for Privacy-ABCs. The particular properties of Privacy-ABCs prevent linking of presentation tokens to the User or across presentation tokens used in different transactions unless the attribute values (or other information out of the scope of the ABCs) allow linking. The privacy of Users is therefore not harmed even if Issuer and Verifier share information with the intention to track behaviour of Users.

Liability: During the issuance part the Issuer may have the obligation to verify the information to be incorporated as attribute values. The Issuer generally is allowed to limit and exclude the liability for such damages.[12] This exclusion must be visible for potential Verifiers, e.g. by publishing the statement of the limitation of liability as part of the terms of service.

[12] This is accepted by even for certification authorities issuing qualified certificates for digital signatures, see Art. 6 para. 3 and para. 3 of Directive 1999/93/EC on a Community framework for electronic signatures. Issuers are allowed to indicate limitations on the use of the certificates and stipulate a limit on the value of transactions for which a certificate can be used. The same must then be accepted for credentials issued and intended for other purposes than securing business transactions.

If the Issuer also acts as Revocation Authority, liability is possible for a damage that occurs in case the Issuer failed to comply with the obligations as a Revocation Authority. In particular not revoking a credential within reasonable time after an issue has been reported, may cause an indemnifiable damage for a Verifier who is relying on an outdated or compromised credential.

6.3 Conclusion and outlook

Privacy-ABCs may not only be object of legal evaluation, but once they are broadly available could change the setting for evaluating other systems. The particular properties of Privacy-ABCs could trigger a reassessment of what must be considered necessary processing for a given purpose. Legal research will need to assess whether and how the deployment of Privacy-ABCs may be made mandatory for data controllers.

As Privacy-ABCs offer a wide range of use cases, it is hard to classify legal relations between the parties involved on an abstract level. However, some general aspects have been analysed and potential legal issues identified. More detailed legal evaluations will follow in the future, based on the use cases of the two pilots to be launched by ABC4Trust. Based on the lessons learnt from the pilots, legal requirements regarding the Inspector and the Revocation Authority will be developed and integrated as important part of the respective pilot descriptions.

7 Applicability to existing Identity Infrastructures

Many identity protocols and frameworks are in use today, and new ones are being developed by the industry, each addressing specific use cases and deployment environments. Privacy concerns exist in many scenarios targeted by these systems, and therefore it is useful to understand how they could benefit from Privacy-ABC technologies to improve their security, privacy, and scalability.

In this chapter, we consider the following popular systems: WS-*, SAML, OpenID, OAuth, and X.509. A short description of each system is given to facilitate the discussion, but is by no means complete; the reader is referred to the appropriate documentation to learn more about a particular system. Moreover, we mostly describe "how" integration can be done, rather than discussing "why" as this is highly application-specific.

The last section describes the common challenges of these federated systems, and how Privacy-ABC technologies can help to alleviate them.

7.1 WS-*

The set of WS-* specifications define various protocols for web services and applications. Many of these relate to security, and in particular, to authentication and attribute-based access (such as WS-Trust [WSTrust], WS-Federation [WSFed], and WS-SecurityPolicy [WSSecPol]). These specifications can be combined to implement various systems with different characteristics.

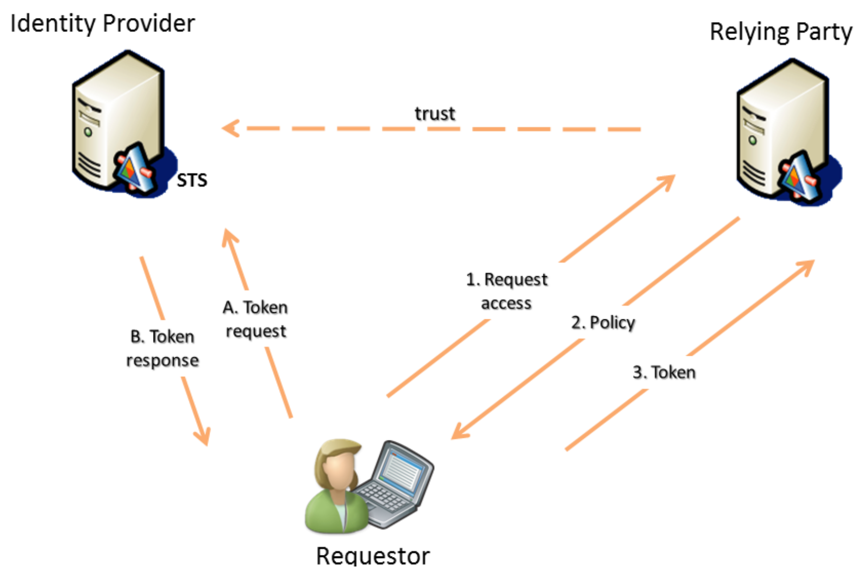

Figure 7.1 - WS-Trust protocol flow

The WS-Trust specification is the main building block that defines how *security tokens* can be obtained and presented by users. The specification does not make any assumption on the type of tokens exchanged, and provides several extensibility points and protocol flow patterns suitable for Privacy-ABC technologies.

In WS-Trust, a requestor (user) requests a security token from the Identity Provider's Security Token Service (the issuer) encoding various certified claims (attributes), and presents it (either immediately or at a later time) to a Relying Party (the verifier); see Figure 7.1.

Integrating Privacy-ABC technologies in WS-Trust is straightforward due to the extensible nature of the WS-* framework. The issuance protocol is initiated by the requestor by sending, as usual, a `RequestForSecurityToken` message to the STS. The requestor and the STS then exchange as many `RequestForSecurityTokenResponse` messages as needed by the ABC issuance protocol (using the challenge-response pattern defined in Section 8 of [WS-Trust]). The STS concludes the protocol by sending a `RequestForSecurityTokenResponseCollection` message. Typically, this final message contains a collection of requested security tokens. Due to the nature of the Privacy-ABC technologies, the STS does not send the security tokens per se, but the requestor is able to compute its credential(s) using the exchanged cryptographic data. See Figure 7.2.

The issuance messages are tied together using a unique context, but otherwise do not specify the content and formatting of their contents. It is therefore possible to directly use the protocol artefacts defined earlier in this document (see Chapter 5).

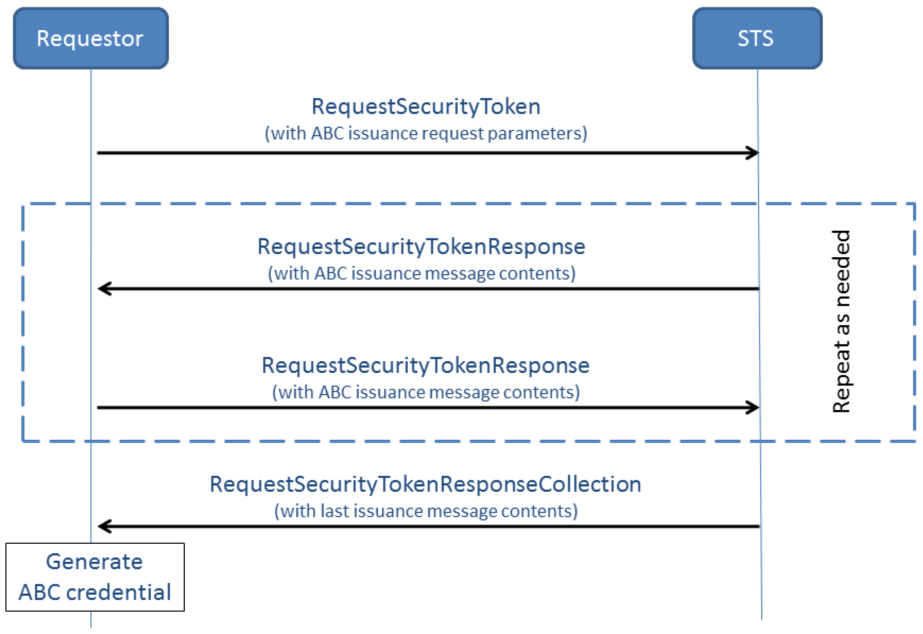
Requestor
STS
RequestSecurityToken
(with ABC issuance request parameters)
RequestSecurityTokenResponse
(with ABC issuance message contents)
RequestSecurityTokenResponse
(with ABC issuance message contents)
Repeat as needed
RequestSecurityTokenResponseCollection
(with last issuance message contents)
Generate ABC credential

Figure 7.2 - WS-Trust issuance protocol

Presenting an ABC to a Relying Party is also straightforward. The exact mechanism to use depends on the application environment. For example, in a federated architecture using WS-Federation, the presentation token could be included in a `RequestForSecurity TokenResponse` message part of a `wresult` HTTP parameter. Given the support of extensible policy (using, e.g., WS-SecurityPolicy), the ABC verifier policy could be expressed by the Relying Party and obtained by the client; e.g., it could be embedded in a service's federation metadata (see Section 3 of [WSFed]).

Privacy-ABC technology integration into WS-Trust has been successfully demonstrated; see, e.g., [UPWTP].

7.2 SAML

The Security Assertion Markup Language (SAML) is a popular set of specifications for exchanging certified assertions in federated environments. Different profiles exist addressing various use cases, but the core specification [SAML2.0] defines the main elements: the SAML assertion (a XML token type that can encode arbitrary attributes), and the SAML protocols for federated exchanges.

Typically, a User Agent (a.k.a. requester or client) requests access to a resource from a Relying Party (a.k.a. Service Provider) which in turn requests a SAML assertion from a trusted Identity Provider (a.k.a. SAML Authority). The User Agent is redirected to the Identity Provider to retrieve the SAML assertion (after authenticating to the Identity Provider in an unspecified manner) before passing it back to the Relying Party. Figure 7.3 illustrates the protocol flow.

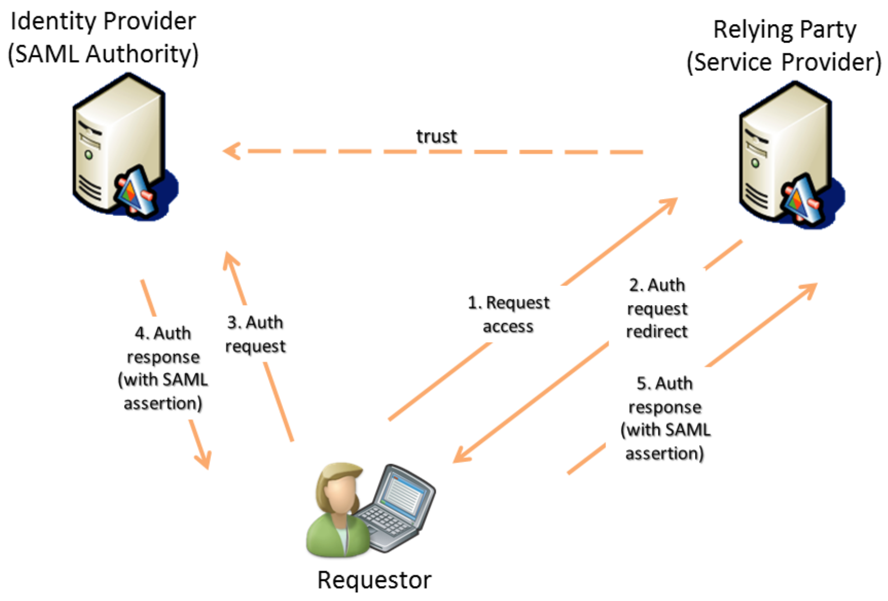
Identity Provider (SAML Authority)
Relying Party (Service Provider)
trust
1. Request access
2. Auth request redirect
3. Auth request
4. Auth response (with SAML assertion)
5. Auth response (with SAML assertion)
Requestor

Figure 7.3 - SAML protocol flow

Contrary to WS-*, the SAML protocols only permit the use of the SAML assertion token type. Therefore, one needs to profile the SAML assertion in order to use the Privacy-ABC technologies with the SAML protocols. The SAML assertion schema defines an optional `ds:Signature` element used by the Identity Provider to certify the contents of the assertion. If used, it must be a valid XML Signature [XMLSignature]. This means that XML Signature must also be profiled to support ABC issuer signatures.[13] The alternative would be to protect the SAML assertion using a custom external signature element.

ABC-based SAML assertions could be used in the SAML protocols in various ways. One example would be for the client to create a modified SAML assertion using a Privacy-ABC in response to a Relying Party's authentication request rather than fetching it in real-time from the Identity Provider (replacing steps 3 and 4 in the figure above). The assertion would contain the disclosed attributes, and encode the presentation token's cryptographic data in the SAML signature. Essentially, the SAML assertion would be an alternative token type to the ABC presentation token.

[13] This could be achieved by applying the appropriate XML transforms on the assertions contents before interpreting them as input to the ABC protocols.

Additionally, the Identity Provider could issue an on-demand Privacy-ABC using the SAML protocol; this might require multiple roundtrips to accommodate the potentially interactive issuance protocol. Then the SAML assertion presented to the Relying Party would need to be created as explained above.

7.3 OpenID

OpenID is a federated protocol allowing users to present an identifier[14] to Relying Parties by first authenticating to an OpenID Provider. The current specification, OpenID 2.0 [OpenID2.0], specifies the protocol. We illustrate the steps in Figure 7.4:

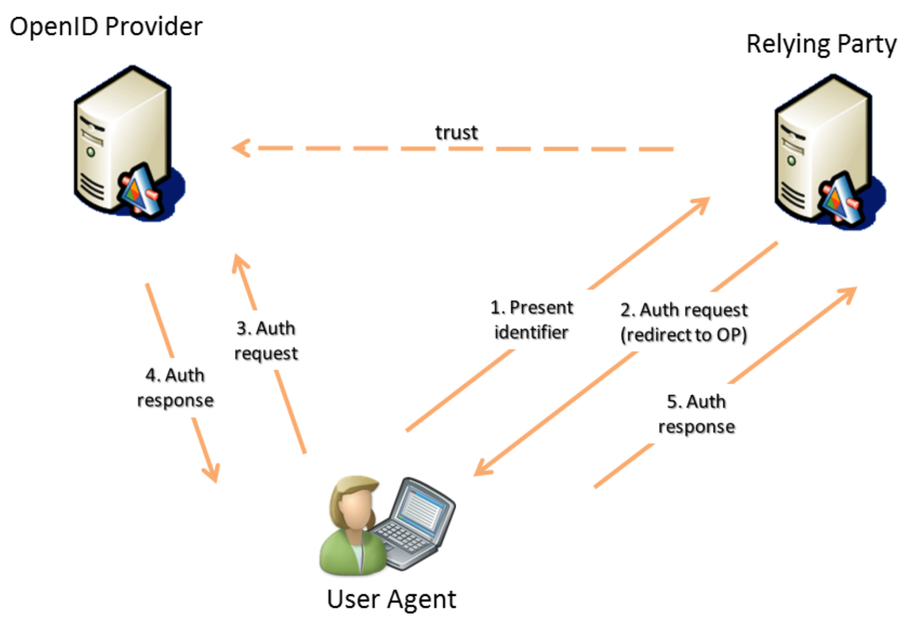

Figure 7.4 - OpenID protocol flow

We assume that the user has an existing OpenID identifier registered with an OpenID Provider.

1. To login to a Relying Party, the user presents her (unverified) OpenID identifier.
2. The Relying Party parses the identifier to discover the user's OpenID Provider and redirects the User Agent to it.
3. The user authenticates to the OpenID Provider; how this is achieved is out-of-scope of the OpenID specification (popular existing web deployments use usernames and passwords).
4. Upon successful authentication, the OpenID Provider redirects the User Agent to the Relying Party with a signed successful authentication message.
5. The Relying Party validates the authentication message using either a shared secret with the OpenID Provider or alternatively, by contacting the OpenID Provider directly.

OpenID follows a standard federated single sign-on model and therefore inherits the security and privacy problems of such systems. The OpenID specification describes in Section 15 some countermeasures against common concerns, but nonetheless, the systems remains vulnerable to active

[14] the specification describe this as a URL or XRI (eXtensible Resource Identifier), but extensions used by popular deployments use email addresses.

attackers, especially to attacks originating from protocol participants (see, e.g., [IDCorner] for a summary of the issues).

Privacy-ABC technologies could be used to increase both the security and privacy of the protocol, and reduce the amount of trust needed on OpenID Providers. For example, certified or scope-exclusive pseudonyms derived from an ABC issued by an OpenID Provider could be used as local Relying Party identifiers, therefore providing unlinkability between the user's spheres of activities at different Relying Parties (using the Relying Partie's URL as a scope string). The cryptographic data in the corresponding ABC presentation token would need to be encoded in extension parameters defined in an ABC profile.

A similar integration has been demonstrated in the PseudoID prototype [PseudoID], using Chaum's blind signatures [Cha82].

OpenID may also be used in attribute-based access scenarios. The OpenID Attribute Exchange [OIAE1.0] extension describes how Relying Party can request attributes of any type from the OpenID Provider by adding fetch parameters in the OpenID authentication message, and how an OpenID Provider can return the requested attributes in the response.

To generate an ABC-based response, the User Agent would create the OpenID response on behalf of the OpenID Provider using the contents of a presentation token, properly encoding the disclosed attributes using the OpenID Attribute Exchange formatting and by encoding the cryptographic evidence in custom attributes.

7.4 OAuth

OAuth is an authorization protocol that enables applications and devices to access HTTP[15] services on behalf of users using delegated tokens rather than the users' main credentials. The current specification, OAuth 1.0, is specified in RFC 5849 [OAuth1.0].

The OAuth 2.0 [OAuth2.0] is now being developed by the IETF OAuth working group.[16] This new version simplifies the base protocol and defines multiple profiles adapted for different scenarios. We will concentrate our discussion on this upcoming standard.

OAuth specifies four roles. Quoting from the spec:

- **resource owner**: an entity capable of granting access to a protected resource (e.g. end-user).
- **resource server**: the server hosting the protected resources, capable of accepting responding to resource requests using access tokens.
- **client**: an application making protected resource requests on behalf of the owner and with its authorization.
- **authorization server**: the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

An example scenario is as follows: an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server) which issues the service delegation-specific credentials (access token).

A typical OAuth interaction is illustrated in Figure 7.5:

[15] Using a transport protocol other than HTTP is undefined by the specification.
[16] OAuth 2.0 evolved from the OAuth WRAP [OAuthWRAP] profile which has been deprecated.

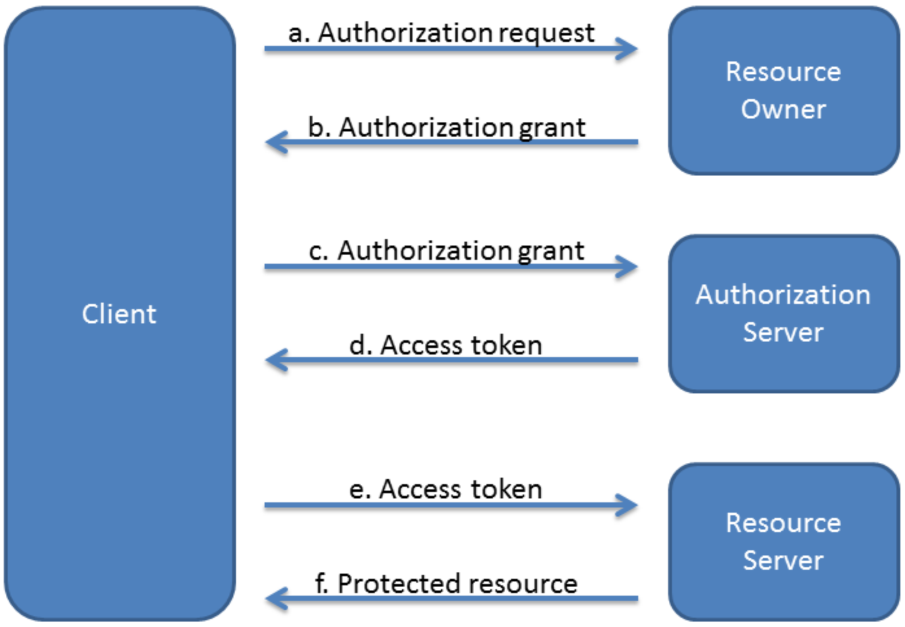

Figure 7.5 - OAuth 2.0 protocol flow

a. The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary.
b. The client receives an authorization grant which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.
c. The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
d. The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token.
e. The client requests the protected resource from the resource server and authenticates by presenting the access token.
f. The resource server validates the access token, and if valid, serves the request.

As we can see, two types of credentials are used in the protocol flow: the authorization grant and the access token. A Privacy-ABC could be used for either one, as we will describe in the following sections.[17] The OAuth protocol flow does not allow presenting a dynamic policy to the client; if this functionality is needed, the policy would need to be obtained and processed at the application layer; otherwise, the application may use an implicit policy that drives the client's behaviour.

[17] The OAuth specification does not describe how the resource owner authenticates the client before issuing the authorization grant. Conceptually, this could also be done using an ABC.

7.4.1 Authorization grant

The first step in the OAuth flow is for the client to request authorization from the resource owner and getting back an authorization grant. The OAuth specification defines four grant types (authorization code, implicit, resource owner password credentials, and client credentials) and provides an extension mechanism for defining new ones.

Although one could use the authorization code or the client credential grant types, the extension mechanism is better-suited to integrate ABC-based grants. How the Privacy-ABC is obtained by the client is out-of-scope of the OAuth flow. To present the Privacy-ABC to the authorization server, one could define a profile similar to the SAML assertion one [OAuthSAML2]. For example, the client could send the following access token request to the authorization server:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=http://abc4trust.eu/oauth&abctoken=PEFzc2VydGlvbiBJc
3N1ZUluc3RhbnQ9IjIwMTEtMDU[...omitted for brevity...]aG5TdGF0Z
```

where the `abctoken` parameter would contain an encoding of a presentation token (e.g., using a base64 encoding of the XML representation). As mentioned above, the policy driving the client's presentation behaviour would be dealt with at the application level (and might be fixed for an application).

7.4.2 Access token

An access token is issued by the authorization server to the client and later presented to the resource server. The format and contents of the access token is not defined in the OAuth specification, and therefore one could define a way to use a Privacy-ABC to create an access token. This can be done by defining a new access token type (as explained in Section 8.1 of [OAuth2.0]), or by encoding the presentation token content into an existing extensible token type, such as the JSON Web Token [JWT].[18]

Since access tokens are typically long-lived, the issuance of the Privacy-ABC can be done out-of-band of the OAuth protocol. It can also be done directly by the authorization server by embedding the issuance protocol messages in multiple access token request-response runs (in which case the returned "access tokens" would be the opaque issuance messages). When this process concludes, the client would be able to create a valid ABC-based access token.

To present the ABC access token, client computes a valid presentation token using an application-specific resource policy (obtained out-of-band or implicitly defined), encodes it in the right access token format, and includes it in the OAuth protected resources access request.

7.5 X.509 PKI

Most of the schemes presented in this chapter require online interactions with an Issuer to present attributes to a Relying Party. This provides flexibility about what can be disclosed to the Relying Party, but impacts the privacy vis-à-vis the Issuer (which typically learns where the attributes are presented). A Public Key Infrastructure (PKI) uses a different approach: PKI certificates encoding

[18] The JSON Web Token format contains a set of attribute name and value pairs and corresponding metadata (including a digital signature identified by an algorithm identifier). This is supported by ABC technologies, but does not allow the representation of the most advanced features.

arbitrary attributes and issued to users are typically long-lived. The decoupling of the issuance and presentation protocols provides some privacy benefits to the user, but removes the minimal disclosure aspect. Indeed, a Verifier will learn everything that is encoded in a certificate even if a subset of the information would have been sufficient to make its access decision. The integration of Privacy-ABC technology is therefore desirable to provide these privacy benefits while offering the same security level as in PKI.

X.509 [X.509] is a popular PKI standard[19] that defines two types of credentials: public key and attribute certificates. A public key certificate contains a user public key associated to a secret private key, and other metadata (serial number, a validity period, a subject name, etc.) The certificate is signed by a Certificate Authority. An attribute certificate, also signed by the CA, is tied to a public-key certificate and can contain arbitrary attributes. Both types of certificates can also contain arbitrary extensions.

The X.509 protocol flow is as follows. The client starts by generating a key pair, and sends a certificate request that includes the generated public key to the Certificate Authority. The Certificate Authority creates, signs and returns the X.509 certificate to the client which stores it along with the associated private key. To authenticate to a Relying Party, the client later uses the certificate's private key to sign a Relying Party-specified challenge (either a random number or an application-specific message). The Relying Party verifies the signature and validates the certificate. This involves verifying the certificate's Certificate Authority signature, making sure that the Certificate Authority is a trusted issuer (is or is linked to a trusted root), and making sure that the certificate has not expired and is not revoked. Checking for non-revocation can be done by either checking that the certificate's serial number does not appear on a Certificate Revocation List (CRL), or by querying an Online Certificate Status Protocol (OCSP) responder.[20] See Figure 7.6.

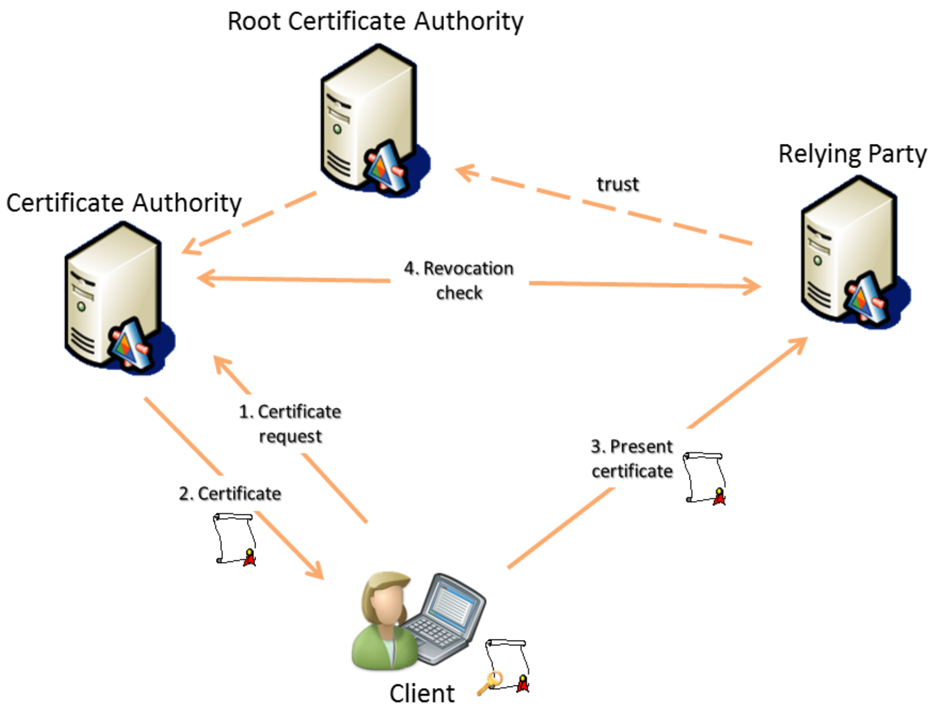

Figure 7.6 - X.509 protocol flow

[19] Other PKI systems exist, such as PGP [PGP]. We will not consider them in this document, but ABC integration would look similar.
[20] The mechanism and endpoint to be used are specified by the CA and encoded into the certificate.

Integrating Privacy-ABCs with X.509 certificates is possible and provides two immediate benefits:

- Long-lived certificates support minimal disclosure (only the relevant properties of encoded attributes are disclosed to the Relying Party rather than the full set of attributes), and
- The user's public key and the Certificate Authority signatures on the certificates are unlinkable (the Certificate Authority and the Relying Parties cannot track and trace the usage of the certificate based solely on these cryptographic values).

Two integration approaches are considered next. The first one consists of encoding the ABC artefacts' contents in X.509 artefacts using ABC-specific algorithm identifiers and extensions (i.e., the client would generate an X.509 certificate encoding the Privacy-ABC's contents at the end of the issuance protocol). Since the presentation protocol of an X.509 certificate is not specified, the presentation token artefact could be used almost as is, but including the modified X.509 certificate.

The second and preferred[21] approach would be to transform an existing X.509 certificate into a Privacy-ABC that can be presented to various Relying Parties. The following example illustrates the concept: The protocol flow would be as follows:

1. The client visits the ABC issuer and presents her X.509 certificate.
2. After validating the certificate and its ownership by the User, the ABC Issuer issues a Privacy-ABC encoding the certificate's information into attributes:
    a. The certificate's expiration date is encoded in an attribute.
    b. The certificate's serial number is encoded as the revocation handle.
    c. The revocation information (e.g., the CRL endpoint)[22] is encoded in an attribute.
    d. The Certificate Authority identifier is encoded in an attribute.
    e. The other certificate fields might also be encoded in the Privacy-ABC if they need to be presented to Relying Parties.
3. The client later presents the ABC to the Verifier, disclosing the following information:
    a. Disclose the Certificate Authority identifier[23] and revocation information attributes.
    b. Prove that the underlying certificate is not expired by proving that the undisclosed expiration date is not before the current time.
    c. Prove that the serial number does not appear on the current CRL (this can be achieved using repetitive negation proofs on the CRL elements).[24]
4. The Verifier would perform these validation steps (on top of the normal ABC validation):
    a. Verify that the Certificate Authority is from a trusted set of issuers.
    b. Retrieve the current CRL (using the disclosed revocation information) and verify the non-revocation proof.
    c. Verify the non-expiration proof.

After these steps, the Verifier is convinced that the user possesses a valid (i.e., non-expired, non-revoked) X.509 certificate from a trusted Certificate Authority.

[21] We claim that this approach is preferred because of the broad existing code base implementing X.509. It would be easier to develop an conversion module on top of existing X.509 components.
[22] This example uses a CRL as the revocation mechanism. Using OCSP would also be possible by having the client prove to the OCSP responder directly that the ABC is not revoked, and presenting a freshly issued "receipt" to the Relying Party.
[23] Alternatively, the client could prove that the CA is from a trusted set specified by the Verifier.
[24] Alternatively, an ABC Revocation Authority could create an accumulator for the revoked values.

7.6 Integration summary

The systems presented above follow a similar federated pattern of a Relying Party requesting, through the user, login or attribute information from a trusted Identity Provider. In PKI and OAuth the certified information (certificate and access token, respectively) are typically obtained in advance and reused over time, while in the other systems, the information is retrieved on-demand from the Identity Provider.

These architectures have some security, privacy, and scalability challenges that might be problematic in some scenarios:

- The Identity Provider can often access the Relying Party using a user's identity without the user's knowledge. This is trivial in systems where the Identity Provider creates the pseudonym (like in SAML, OpenID, OAuth, WS-Federation). In systems where a user secret is employed (like in PKI, or in some WS-Trust profiles), this is more complicated but still could be possible.[25] Moreover, Identity Providers can also selectively deny access to users by refusing to issue security tokens (discriminating on the requesting user or requested service).
- For authentication depending on knowledge of a user secret (e.g., username/password), phishing attacks on the credential provided to the Identity Provider result in malicious access to all Relying Parties that accept that identity.
- Strong authentication to the Identity Provider is often supported (including multi-factor asymmetric-based authentication), but the resulting security tokens (e.g., SAML assertion, OAuth access token, OpenID authentication response) are typically weaker software-only bearer token which can be intercepted and replayed by adversaries.
- The Identity Provider typically learns which Relying Party the user is trying to access. For on-demand security token issuance, this information is often provided to the Identity Provider in order to protect the security token (e.g., to encrypt it for the Relying Party) or to redirect the user to the right location. When security tokens are long-lived (like in PKI), this information is still available if the Identity Providers and Relying Parties compare notes (since signatures on security tokens generated using conventional cryptography are traceable).
- Central Identity Providers in on-demand federated systems limit the scalability of the systems because if they are offline, users will not be able to access any Relying Parties. This makes them interesting targets for denial of service attacks.

Privacy-ABC technologies help alleviate these issues by increasing the security, privacy, and scalability of these systems. Indeed:

- Since Privacy-ABCs are by default untraceable, even when obtained on-demand, Identity Providers are not able to track and trace the usage of the users' information.
- Since Privacy-ABCs can be obtained in advance and stored by the user while still being able to disclose the minimal amount of information needed for a particular transaction, the real-time burden of the issuer is diminished, improving scalability.
- Since Privacy-ABCs are based on asymmetric cryptography, presenting login pseudonyms and certified attributes involve using a private key unknown to the Issuer, meaning that the

[25] As an example, in PKI, a Certificate Authority would not be able to re-issue a valid certificate containing the user's public key, but could re-issue one with a matching serial number and subject and key identifiers often used for user authentication.

Identity Provider (or another adversary) is unable to hijack the user's identity at a particular Relying Party.

Privacy-ABC technologies offer a wide range of features; not all of them trivially compatible with the systems presented in this chapter. The important point is that Privacy-ABC technologies offer a superset of the functionality and of the security/privacy/scalability characteristics of these systems. Protocol designers and architects can therefore pick and choose which features and characteristics they would like to use to improve existing systems or their future revisions.

It is also important to note that Privacy-ABC technologies can be used in conjunction with these frameworks, since many real-life applications won't have the luxury to modify the existing standards and development libraries. Most of the privacy concerns occur in cross-domain data sharing, i.e., when information travels from one domain to another. Therefore, an ABC "proxy" can be used as a privacy filter between domains using well-known federated token transformer pattern (such as the WS-Trust STS). This is useful to avoid modifying legacy applications and infrastructure, and still benefit from the security and privacy properties of Privacy-ABC technologies.

8 Glossary

Attribute

A piece of information, possibly certified by a credential, describing a characteristic of a natural person or entity, or of the credential itself. An attribute consists of an attribute type determining the semantics of the attribute (e.g., first name) and an attribute value determining its contents (e.g., John).

Certified pseudonym

A verifiable pseudonym based on a user secret that also underlies an issued credential. A certified pseudonym is established in a presentation token that also demonstrates possession of a credential bound to the same User (i.e., to the same user secret) as the pseudonym.

Credential

A list of certified attributes issued by an Issuer to a User. By issuing a credential, the Issuer vouches for the correctness of the contained attributes with respect to the User.

Credential specification

A data artifact specifying the list of attribute types that are encoded in a credential.

Data Controller

“'Controller' shall mean the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data...”, Art. 2 (d) of Directive 95/46/EC. In the area of Privacy-ABCs the Issuer, Verifier, the Revocation Authority and the Inspector are Data Controllers with the respective duties arising from the law.

Data Processor

“'Processor' shall mean a natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller“, Art. 2 (e) of Directive 95/46/EC. Data Controllers processes personal data on behalf of the data Controller.

Data Subject

A data subject is an identified or identifiable natural person, Art. 2 (a) of Directive 95/46/EC. In the area of Privacy-ABCs the User and any other national person of which personal data is processes is a data subject. Data subjects have data subjects’ rights assigned such as the right of access, rectification, erasure and blocking, Art. 12 of Directive 95/46/EC.

Device binding

An optional credential feature whereby the credential is bound to a strong secret embedded in a dedicated hardware device so that any presentation token involving the credential requires the presence of the device.

Inspection

An optional feature allowing a presentation token to be de-anonymized by a dedicated Inspector. At the time of creating the presentation token, the User is aware (through the presentation policy) of the identity of the Inspector and the valid grounds for inspection.

Inspection grounds

The circumstances under which a Verifier may ask an Inspector to trace the User who created a given presentation token.

Inspection Requester

Entity requesting an inspection from the Inspector, asserting that inspection is compliant with the inspection grounds specified or is legally required. In most cases this will be the Verifier, but also may be the police, or other legally authorised entity.

Inspector

A trusted entity that can trace the User who created a presentation token by revealing attributes from the presentation token that were originally hidden from the Verifier.

Issuance key

The Issuer's secret cryptographic key used to issue credentials.

Issuer

The party who vouches for the validity of one or more attributes of a User, by issuing a credential to the User.

Issuer parameters

A public data artifact containing cryptographic and other information by means of which presentation tokens derived from credentials issued by the Issuer can be verified.

Linkability

See *unlinkability*.

Personal data

"'Personal data' shall mean any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity", Art. 2 (a) of Directive 95/46/EC. Within this deliverable personal data is the terminology used for legal considerations. See also *Personally Identifiable Information*.

Personally Identifiable Information (PII)

Personally Identifiable Information is defined as any information about an individual maintained by an [entity], including any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, and any other information that is linked or linkable to an individual ([NIST10] p. 2-1). PII is a widely used terminology for *personal data* in the domain of information security. Within this document PII is used in relation to information security.

Presentation policy

A policy created and published by a Verifier specifying the class of presentation tokens that the Verifier will accept. The presentation policy contains, among other things, which credentials from which Issuers it accepts and which information a presentation token must reveal from these credentials.

Presentation token

A collection of information derived from a set of credentials, usually created and sent by a User to authenticate to a Verifier. A presentation token can contain information from several credentials, reveal attribute values, prove that attribute values satisfy predicates, sign an application-specific message or nonce or support advanced features such as pseudonyms, device binding, inspection, and revocation. The presentation token consists of the presentation token description, containing a technology-agnostic description of the revealed information, and the presentation token evidence, containing opaque technology-specific cryptographic parameters in support of the token.

Pseudonym

See *verifiable pseudonym*.

Pseudonym scope

A string provided in the Verifier's presentation policy as a hint to the User which previously established pseudonym she can use, or to which a new pseudonym should be associated. A single User (with a single user secret) can generate multiple verifiable or certified pseudonyms for the same scope string, but can only generate a single scope-exclusive pseudonym.

Revocation

The act of withdrawing the validity of a previously issued credential. Revocation is performed by a dedicated Revocation Authority, which could be the Issuer, the Verifier, or an independent third party. Which Revocation Authorities must be taken into account can be specified by the Issuer in the issuer parameters (Issuer-driven revocation) or by the Verifier in the presentation policy (Verifier-driven revocation).

Revocation Authority

The entity in charge of revoking credentials. The Revocation Authority can be an Issuer, a Relying Party, or an independent entity. Multiple Issuers or Verifiers may rely on the same Revocation Authority.

Revocation information

The public information that a Revocation Authority publishes every time a new credential is revoked or at regular time intervals to allow Verifiers to check that a presentation token was not derived from revoked credentials.

Revocation parameters

The public information related to a Revocation Authority, containing cryptographic information as well as instructions where and how the most recent revocation information and non-revocation evidence can be obtained. The revocation parameters are static, i.e., they do not change every time a new credential is revoked or at regular time intervals like the revocation information and non-revocation evidence (may) do.

Non-revocation evidence

The User-specific or credential-specific information that the user agent maintains, allowing it to prove in presentation tokens that the credential was not revoked. The non-revocation evidence may need to be updated either at regular time intervals or when new credentials are revoked.

Scope

See *pseudonym scope.*

Scope-exclusive pseudonym

A certified pseudonym that is guaranteed to be cryptographically unique per scope string and per user secret. Meaning, from a single user-bound credential, only a single scope-exclusive pseudonym can be generated for the same scope string.

Traceability

See *untraceability*.

Unlinkability

The property that different actions performed by the same User, in particular different presentation tokens generated by the same User, cannot be linked to each other as having originated from the same User.

Untraceability

The property that an action performed by a User cannot be traced back to her identity. In particular, the property that a presentation token generated by a User cannot be traced back to the issuance of the credential from which the token was derived.

User

The human entity who wants to access a resource controlled by a verifier and obtains credentials from Issuers to this end.

User agent

The software entity that represents the human User and manages her credentials.

User binding

An optional credential feature whereby the credential is bound to an underlying user secret. By requiring multiple credentials to be bound to the same user secret, one can prevent Users from “pooling” their credentials.

User secret

A piece of secret information known to a User (either a strong random secret or a human-memorizable password or PIN code) underlying one or more issued credentials or pseudonyms. A presentation token involving a pseudonym or a user-bound credential implicitly proves knowledge of the underlying user secret.

Verifiable pseudonym

A public identifier derived from a user secret allowing a User to voluntarily link different presentation tokens created by her or to re-authenticate under a previously established pseudonym by proving knowledge of the user secret. Multiple unlinkable pseudonyms can be derived from the same user secret.

Verifier

The party that protects access to a resource by verifying presentation tokens to check whether a User has the requested attributes. The Verifier only accepts credentials from Issuers that it trusts.

9 Acronyms

ABCs

Attribute Based Credentials

ABCE

ABC Engine

CA

Certificate Authority

CE

Crypto Engine

ENISA

European Network and Information Security Agency

FP7

Framework Programme 7

HTTP

Hypertext Transfer Protocol

HTTPS

HyperText Transfer Protocol Secure (HTTP secured by TLS or SSL)

ID

Identifier

Idemix

IBM Identity Mixer

ICT

Information and Communications Technology

IdM

Identity Manager

ISO

International Organisation for Standardisation

IdSP

Identity Service Provider

PET

Privacy Enhancing Technology

PRIME

Privacy and Identity Management for Europe

PrimeLife

	Privacy and Identity Management in Europe for Life
PIN	Personal Identification Number
RP	Relying Party
SCI	Smartcard Interface
SSL	Secure Sockets Layer
STS	Secure Token Service
TLS	Transport Layer Security
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

10 Bibliography

[Art29WP114] Article 29 Working Party, *Working document on a common interpretation of Article 26(1) of Directive 95/46/EC of 24 October 1995*, Adopted on 25 November 2005, online: http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2005/wp114_en.pdf.

[Art29WP118] Article 29 Working Party, *Opinion 15/2011 on the definition of consent* Adopted on 13 July 2011, online: http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2011/wp187_en.pdf.

[Art29WP169] Article 29 Working Party, *Opinion 1/2010 on the concepts of 'controller' and 'processor'* Adopted on 16 February 2010, online: http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2010/wp169_en.pdf.

[BCGS09] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard java card. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 600–610, New York, NY, USA, 2009. ACM.

[BDDD07] Stefan Brands, Liesje Demuynck, and Bart De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. In Proceedings of the *12th Australasian conference on In- formation security and privacy*, ACISP'07, pages 400–415, Berlin, Heidelberg, 2007. Springer-Verlag.

[Bra93] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, pages 302–318, 1993.

[Bra00] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.

[BT08] Deutscher Bundestag, "Drucksache 16/10489: Entwurf eines Gesetzes über Personalausweise und den elektronischen Identitätsnachweis sowie zur Änderung weiterer Vorschriften", Berlin, Germany 2008.

[Cam06] Jan Camenisch. Protecting (anonymous) credentials with the trusted computing group's TPM v1.2. In *SEC*, pages 135–147, 2006.

[CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, pages 234–252, 2008.

[CG08] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In *ACM Conference on Computer and Communications Security*, pages 345–356, 2008.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *ACM Conference on Computer and Communications Security*, pages 201–210, 2006.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *SCN*, pages 141–155, 2006.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography*, pages 481–500, 2009.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72, 2004.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[DEF11] Digital Enlightenment Forum, *The Future of European Electronic Identity Management*, Position Paper, 31 October 2011, Paris, France.

[ENISA09] European Network and Information Security Agency, *Privacy Features of European eID Card Specifications*, Position Paper, February 2009, http://www.enisa.europa.eu/act/it/ privacy-and-trust/ eid/eid-cards-en

[FP11] Walter Fumy and Manfred Paeschke, editors. *Handbook of eID Security: Concepts, Practical Experiences, Technologies*. John Wiley And Sons, February 2011.

[HoSch11] Leif-Erik Holtz, Jan Schallaböck. Legal Policy Mechanisms. In Jan Camenisch, Simone Fischer-Hübner, Kai Rannenberg, editors, Privacy and Identity Management for Life, 343354, Springer Heidelberg, 2011.

[IDCorner] Stefan Brands, The ID Corner blog. The problem(s) with OpenID. http://www.untrusted.ca/cache/openid.html.

[IMI1.0] OASIS Standard. Identity Metasystem Interoperability Version 1.0. 1 July 2009. http://docs.oasis-open.org/imi/identity/v1.0/identity.html

[JWT] JSON Web Token (JWT). draft-jones-json-web-token-05. http://www.ietf.org/id/draft-jones-json-web-token-05.txt

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, pages 275–292, 2005.

[OAuth1.0] The OAuth 1.0 Protocol. http://tools.ietf.org/html/rfc5849

[OAuth2.0] The OAuth 2.0 Authorization Protocol. http://tools.ietf.org/html/draft-ietf-oauth-v2 (draft version 22 at time of writing)

[OAuthSAML2] Mortimore, C., "SAML 2.0 Bearer Assertion Profiles for OAuth 2.0", draft-ietf-oauth-saml2-bearer-08 (work in progress), August 2011.

[OAuthWRAP] OAuth Web Resource Authorization Profiles. January 15, 2010. http://tools.ietf.org/html/draft-hardt-oauth-01

[OIAE1.0] OpenID Attribute Exchange 1.0. December 5, 2007. http://openid.net/specs/openid-attribute-exchange-1_0.html

[OpenID2.0] OpenID Authentication 2.0. December 5, 2007. http://openid.net/specs/openid-authentication-2_0.html

[PseudoID] Arkajit Dey, Stephen Weis. PseudoID: Enhancing Privacy in Federated Login. http://research.google.com/pubs/pub36553.html.

[RFC2119] Scott Bradner. RFC 2119: Key words for use in RFCs to Indicate Requirement Levels, March 1997. http://www.rfc-editor.org/rfc/rfc2119.txt

[SAML2.0] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. 15 March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[UPWTP] U-Prove WS-Trust Profile V1.0. March 2011. http://www.microsoft.com/u-prove.

[WS-Trust14] OASIS Standard. WS-Trust 1.4, February 2nd 2009. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html.

[WSFed] OASIS. Web Services Federation Language (WS-Federation) Version 1.2. OASIS Standard. 22 May 2009. http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html

[WSSecPol] OASIS. WS-SecurityPolicy 1.2. 30 April 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cs.html

[WSTrust] OASIS. WS-Trust 1.4. 2 February 2009. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.html

[X509] ITU-T. X.509 : Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks. http://www.itu.int/rec/T-REC-X.509/en

[XACML20] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 2.0, February 1, 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes Second Edition, 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[XMLSignature] D. Eastlake et al. XML-Signature Syntax and Processing. World Wide Web Consortium, February 2002. http://www.w3.org/TR/xmldsig-core/.

[Zwi11] Harald Zwingelberg. Necessary processing of personal data: The need-to-know principle and processing data from the new german identity card. In Simone Fischer-Hbner, Penny Duquenoy, Marit Hansen, Ronald Leenes, and Ge Zhang, editors, *Privacy and Identity Management for Life*, volume 352 of *IFIP Advances in Information and Communication Technology*, pages 151–163. Springer Boston, 2011.